Virtual Whiteboard to support Scrum

Final project for Pba. Software development

EASV, June 2012.

Student: Yani Kereena Atmakuri Davidsen <yda5001@easv.dk>
Supervisor: Bent H. Pedersen <bhp@easv.dk>
Submitted: 4th of June, 2012.

Table of Contents

Introduction....1
Project Establishment....2
Analysis....6
Design....10
Implementation....12
Discussion....24
Conclusions....31
Acknowledgements....31
References....32
Appendix A: Google Trends analysis....33
Appendix B: ScrumDo documentation....36
Appendix C: User evaluation....39
Appendix D: User manual....41

Introduction

Background of this project is that I am doing project for an Indian outsourcing startup company www.kokila.eu, which will deal with outsourced projects from Europe. While KOKILA has been doing projects for RE-Data in Copenhagen, they found the following challenges related to managing outsourced projects:

1. Agile outsourcing, customers expect agile outsourcing, where they don't have to specify tasks in high detail.
2. Distance: time-zones are different, communication quality may not be good.
3. Culture difference, not sure participants understand each other.

KOKILA would like to use a communication system, which is making it easy for product owner(s) and developer(s) to work together. Customers and consultants are far away from KOKILA developers, but they would still like to know the status of projects and discuss what is going on, as if they were in the same room as developers. Even though there are some project management tools available, they are too complex and not a good match for the outsourcing problems.

Problem Definition

Following our above discussion, I am now able to formulate a set of questions which I will try to answer during the project:

1. *What are the major problems areas in agile software development related to outsourcing?*
2. *What are the solutions can be proposed to build the system that handles specific problems related to agile with offshore?*
3. *Evaluate what is the user's feedback related to the prototype of the system built?*

Project Establishment

In this section I will establish my project. The purpose of this is to get a clear view of what kind of project I am facing and how I should progress with it. Topics are such as definition of goals, analysis of strategy, risks and stakeholders. The outcome is that I have selected a process model for how to approach the project, and I have planned milestones for progress.

Goals

1. Identify the major problems which are related to agile system with outsourcing.
2. Identify what is most important problem to company and that is realistic to make in given time for bachelors project .
3. Complete the project establishment, and selecting process model together with customer within 2 weeks for agreed system to develop.
4. Produce a prototype of a system to implement solutions to problems identified in given time.
5. Evaluate the prototype with real users.

Strategy Analysis

Strategy analysis is a way to find strategic elements to include in the project. By looking at the conditions of the project, and the strengths and weaknesses of those, it is possible to find their strategic solutions. Like with risk analysis, it is a way to get a better understanding of the project, to anticipate future problems with more ease. Below are the strategic conditions I have found for the project:

Stakeholders

First, I try to identify the stakeholders, I have done this using brainstorming.

Stakeholder	Description
Developer	Interested in brainstorming together with customer about scope. Interested in task analysis, estimating, designing, coding and testing.
Supervisor	Interested to give guidance to student / developer in project planning, helps to give or refer materials required to project, to give feedback on project progress.

KOKILA	They are interested in the project to work together with developers at find solutions for their scope, requirement specifications and architecture phases. Interested to evaluate project, give feedback and motivate the developer towards goal.

Strengths and weaknesses for the Project

Conditions	Strength	Weakness	Strategic element
Technical issues	Knowledge in HTML5, Java and SQL server.	Not enough knowledge in Javascript and MongoDB	self study of javascript , Get mentoring on MongoDB.
Developers	Only myself ! so no dependencies. Guidance and material available from supervisor to write project report. Not experienced	Going to work myself alone with analysis, design,coding and testing !	Confirm project steps / outline with supervisor. Troubleshoot from internet , if get problems in technical parts.
Supervisors	Got good supervisor to give guidance by email when I need. Stress free :-)	I am staying far away from school, so I can not meet him, to put direct meetings.	Follow the steps from project foundation material, and confirm with my supervisor.
Product owner	Myself and rest of other partners to evaluate the product I am developing.	Requirements are not specific, can change often by product owners!	So not good to use waterfall process model.
Users	Scrum team in KOKILA is going to use this product in future.	only one customer and developer to use or evaluate product in developing phase	Work close with customer(s) to show prototype to evaluate in each phase.
Environment	No technical problems and no disturbance, peaceful environment at work.	staying far away in another city from school, difficult meet teachers / supervisor.	internet available at work, I can contact Supervisor / teachers by email,if I need help.

Risk Analysis

In the risk analysis area, I can try to identify what are the major obstacles I can encounter in the project, so I can plan around them. I am aware that the nature of the problem is to identify the unexpected risks and risks those are not controllable. Risk is a combination of the severity and probability of the hazard event. This means that by analyzing and categorizing the risks, I get a better understanding , probably some risks I can not avoid but still I can try to minimize.

Risk	Probability	Effect	Acceptability	Solutions
Customers scope and specifications are too abstract.	High	Medium	Acceptable	Take more time to work together with customer to find exact requirements.
May be take too much time for brainstorming together with customers to find requirement specifications to start implementation.	High	High	Intolerant	I do not know how to control it ! Hope, I can try to find solution for it together with customers.
Project may be overrun by unrealistic schedules.	Medium	High	Intolerable	Use time management techniques
Developer do not have experience	high	medium	acceptable	ask company for mentor!
Developer can not estimate accurate time for whole product development !	high	medium	acceptable	Decompose the requirements and do the iterative way and find approximate time for each iteration.
Requirements can change often.	medium	High	intolerable	Discuss and agree with customers about requirement changes before implementation.
System is not user friendly	High	High	Intolerable	Test together with customer to improve quality.

Choosing Process model for the project

After Strategy analysis and following strategic elements, there is high uncertainty in Project. I have to work myself as customer, as developer and also work together with rest of product owners to come out of uncertainty and to find what exact I need to develop. So I would like to choose **Agile process model** for this project which supports iterative and incremental product development and working close together with customer.

Method to develop product

By risk analysis , I would like to use **Scrum** method , because there is great uncertainty about requirements and customers have very abstract scope, since I am alone to implement and missing teamwork inspiration and there are some new technology KOKILA developers suggesting to use. So I need motivation and mentoring to learn new technology such as MongoDB and Javascript etc. ,. By choosing Scrum I can get Scrum master from KOKILA to mentor and control Scrum process and I can work together with customers to minimize the uncertainty. Since customer can not give very specific functional requirements in starting of process and they would like to add new functionalities when they get ideas. So by choosing Scrum I can implement iteratively and incrementally by sprints to support adding new functions and react customers feedback.

Iteration planning

Now I can plan the project schedule with KOKILA and Scrum Master Søren. We decided on sprints (iterations) from one to two weeks duration, and a Scrum daily meeting every 2nd day: Monday, Wednesday and Friday. We decided to use ScrumDo system to manage the Scrum process and artifacts since KOKILA is already using it.

The outcome from the iterations has been included in Appendix B.

Project Schedule & Review

Tasks & User Stories	**iteration**	**Planned time**	**Spent time**	**Status**
Brainstorming with company & project establishment	0	2 weeks (02-04-2012 to 16-04-2012)	2 weeks (02-04-2012 to 16-04-2012)	Finished
Analysis	1	2 weeks (16-04-2012 to 30-04-2012)	2 weeks (16-04-2012 to 30-04-2012)	Finished
Analysis, design, Implementation & testing (creating virtual whiteboard with sharing feature)	2	1 week (30-04-2012 to 07-05-2012)	1 week (30-04-2012 to 07-05-2012	Finished

Analysis,Design, Implementation & testing (related to GUI features)	3	2 weeks (07-05-2012 to 21- 05-2012)	2 weeks (07-05-2012 to 21- 05-2012	Finished
Analysis, Design, Implementation & testing, User evaluation and report writing.	4	2 weeks (21-05-2012 to 04-06-2012)	2 weeks (21-05-2012 to 04-06-2012	User evaluation and project report has been completed. two of the user stories not completed..

Note: In 4.th iteration user stories 2 out of 6 could not implemented in lack of time, (i.e user story-9. Saving snapshots in whiteboard and user story-25). Probably I could build them in 5.th iteration with 3 weeks Sprint in future.

As a note to the reader, the remainder of the report is a summarized output of the iterations done using the Scrum process, and does not follow progress of each iteration. For this, see the Appendix B, as mentioned. For example, the “Analysis” section below was incrementally written during the first iteration, as part of 4 analysis-related stories. Or, the implementation section on “JSON encoding” was done during implementation of a story on communication with client-server in the 2nd iteration.

Analysis

In the analysis phase, I want to know about the problem I am facing. An introduction to Scrum is given and discussion about problems in relation to outsourcing. Here both literature is included and a brainstorming session with KOKILA. Together they give me input for which problem to address when developing a system for KOKILA.

The analysis was primarily done during my first iteration.

Understanding Scrum and outsourced project models

Recent years has shown a great need for outsourced project development, due to financial crisis and high availability of skilled low-cost software developers in e.g. India. At the same time, new types of projects will be outsourced, namely more knowledge-intensive projects, which are less specified from the customer side and require more domain-understanding from the developer side. Combined, this can be identified as a need for the use of agile project models in outsourcing. One such very popular agile project model is Scrum. This section describes the Scrum project model, distributed project development and describes problems related to implementing the two in a software development company (here KOKILA).

Scrum project model

Scrum is a framework for the software development process. It is a framework in the sense that it defines overall activities, but, without going into the details. The philosophy is that the team

knows best how to fill in the details. For example, Scrum defines that the outcome of a “sprint planning” meeting is the commitment of developers, not which documents must be produced, as with other models. Figure 1 shows an overview of the components of an iteration in the Scrum project model. The roles in Scrum consists of the following: 1) product owner -- a person who represents the product and who has authority to make decisions regarding the product; 2) developer team -- the team of developers; 3) scrum master -- a facilitator to make sure the process is smooth. There is no project leader or the like in Scrum, all members are considered equal, but with different responsibilities.

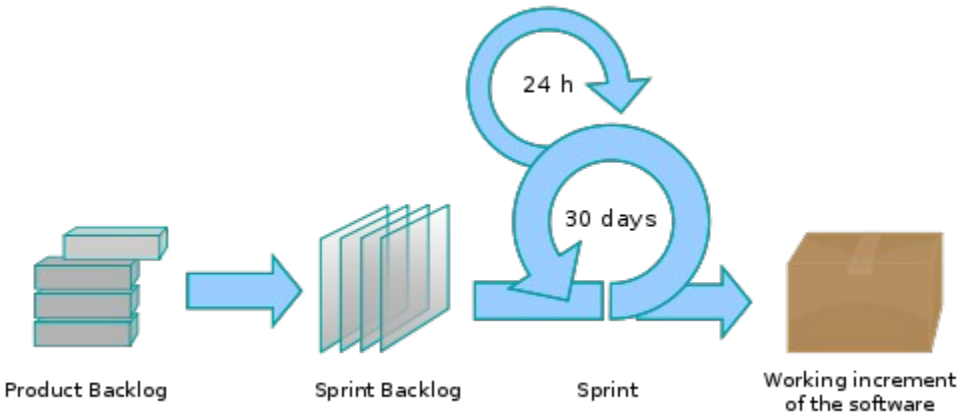


Figure 1: Scrum process overview. Source: Wikipedia.org

In Scrum, the iteration is called a “sprint”. The sprint starts with creating a “sprint backlog” from the “product backlog”. The product backlog are all the desired features of the product owner. To create the “sprint backlog” is done at a “sprint planning” meeting, in which the product owner and developer team prioritize and estimate items from the product backlog. The outcome is a commitment from the developer team, to complete a specific set of items, called the “sprint backlog”. After these items have been selected, the sprint is started, and developers work on the committed items. Each day, a status-meeting is conducted together with the product owner, to get feedback on status and other problems. As mentioned above, during design and implementation, the developers are encouraged to discuss with product owner as often as needed to concretize the items into a piece of software. This is also the outcome of the Sprint: A new working version of the software. If the commitment is not met by the developer team, tasks have to be put back into the product backlog, as well as new items can be added to the product backlog.

A “burndown-chart” is another central piece in Scrum, where the current output of the project is seen, and can be used to forecast/predict progress. Figure 2 shows an example burndown chart.

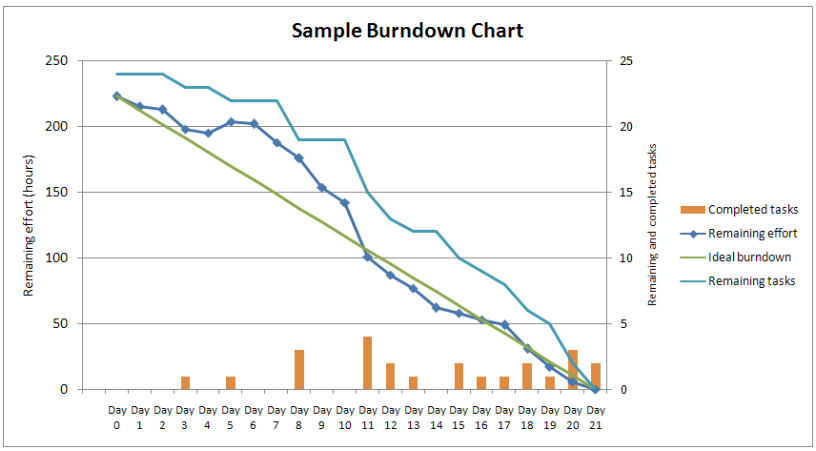


Figure 2: Sample burndown chart for a Sprint. Source: Wikipedia.org

The burndown chart shows remaining effort, here measured in hours but generally “units” are used, as a function of time. One task of the Scrum-master is to distribute this chart to all project members to visualize a generally decreasing data towards the end of the Sprint.

After new software is delivered, another Sprint can start, until the product backlog is empty or, in a time-boxed project, the time is over. Central to Scrum is, as with most agile models, the close communication with the customer side, here the product owner.

Problems in Outsourcing

For discussion about problems in outsourcing, several have been identified from literature (using references 1, 2 and 5). The following points are made by the authors:

1. Specifications could be unclear from Product owner, So developers may be make guesses and assumptions which produces the wrong thing in a product .
2. Developers and Product owners can have different views on same thing they discussed by different backgrounds from different cultures.
3. Rich / poor communication between product owner and developers will influence the business value.
4. Product owner do not have knowledge about programming, so set the tasks business point of view.
5. May be developers do not have the enough knowledge of the business, so they suggest customer to develop some tasks which are not efficient to business.

Brainstorming with KOKILA

After brainstorming together with company I identified their problems of agile development in outsourcing. Problems are: Communication management (how to get the most benefit from discussion with product owner) and time management (product owner is in different time zone). Currently, KOKILA is using Google Calendar, ScrumDo and Skype to handle the above mentioned problems.

Discussing stories with product owner

For communication management, company is using Skype for video conferencing, but still there is uncertainty between product owner and rest of Scrum team about discussing product specifications and architecture etc., even though using Skype for video conference but it is not easy to understand what is exact customer mean and what is exact developer mean. It could be language accent or could be way of explaining etc.

So it could be good idea to develop a virtual whiteboard that is possible to access and work on it same time from different browsers, while participating in Skype video conference. So they can explain by drawings or uml diagrams if they need.

ScrumDo and discussion

There is also need to see during discussion (e.g. on whiteboard) what is the user story they are discussing, it could be good idea to make system that pick user story from the ScrumDo through whiteboard. So integration between ScrumDo and Whiteboard is recommended to built.

Unavailable product owner and/or developers

Sometimes Customer/Product owner would like to contact in emergence to discuss about requirements/changes with developer or Developer would like to call customer's phone from his / her computer by using Skype to discuss their doubts in while developing the product.
Same time if developer or customer not available at computer system, Customer / Developer can call to each others Android phone by Skype.

Problem is even though they can participate into Skype meetings by calling to the android mobile phone, they can not see the whiteboard, so it could need a system that displays live whiteboard. Probably that project is going to be long time, which does not fit to my bachelors project. So I made assumption that I can develop a integration tool, so customer/developer can send snapshots from whiteboard to the android phone while they are discussing on skype, that makes easy to understand better. So they would like a system integrating between virtual whiteboard and an Android phone.

Problem solution statement

Following the above, it is clear that communication is very important in Scrum, and in an outsourced project, communication is not so easy because of the distance between product owner and developers. What KOKILA proposed is the "virtual whiteboard" solution, which allows product owner and the developers to share a whiteboard to supplement the discussion on Skype. Specific for this whiteboard is the need for tools to discuss GUI layout and possibly for technical discussion tools such as UML.

Therefore, my problem solution statement is:

To develop a virtual whiteboard which can be shared between locations and which supports elements of GUI and/or UML design.

Design

After deciding on the problem of developing a virtual whiteboard, the design phase can start. Together with KOKILA, a set of Scrum stories were collected. This section will through the stories and discuss their functions. Figure 3 gives a summarized overview of the stories using an UML use case diagram.

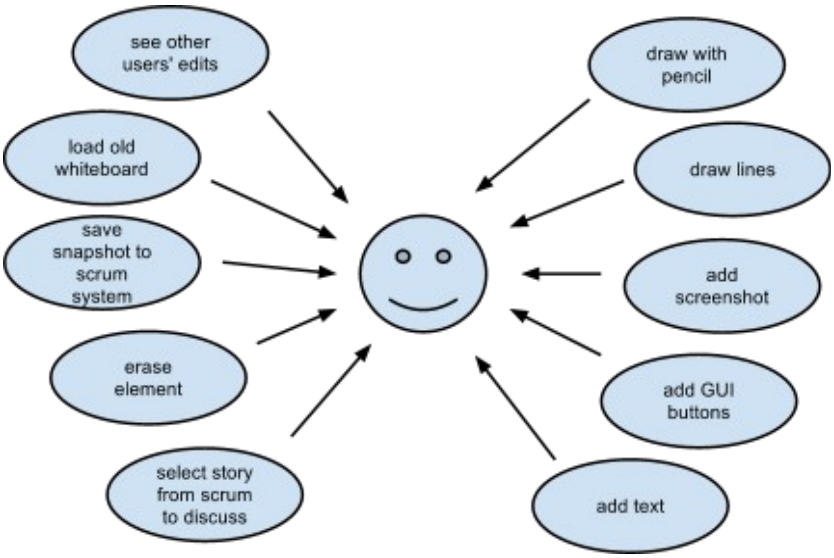
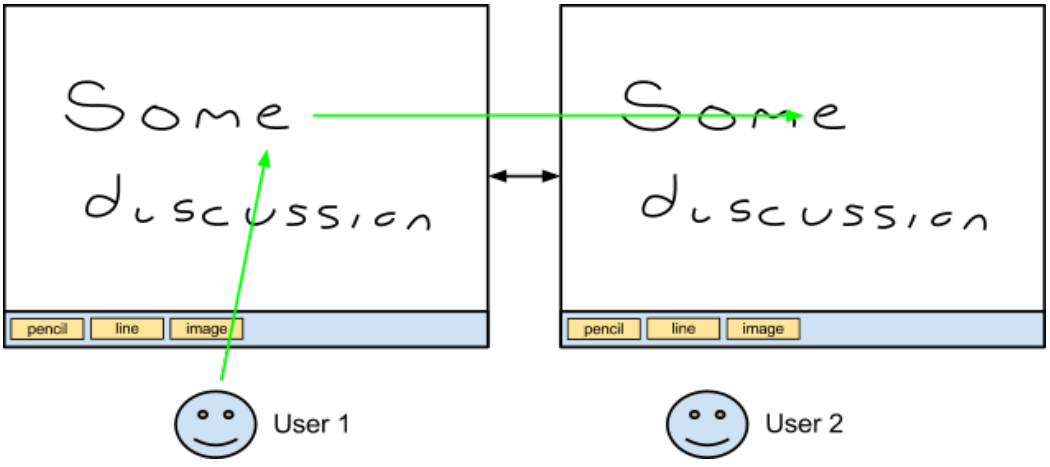


Figure 3: Use Case diagram, overview of stories to be done.

See other users' edits

This story is the requirement that several users should be able to see and edit the whiteboard at the same time in separate locations. When an edit is done in one whiteboard, it should immediately be sent to all other viewers of the whiteboard. Figure 4 shows an example of this.
Figure 4: User 1 draws "e" on the whiteboard, and his edit is immediately shown on User 2's whiteboard.



Load old whiteboard

Whiteboard data should be saved between sessions, so users can come back to a specific whiteboard and continue discussion. Further, it should be able to link directly to a saved whiteboard. Example: http://whiteboard/?whiteboardID=123

Standard drawing tools should be available

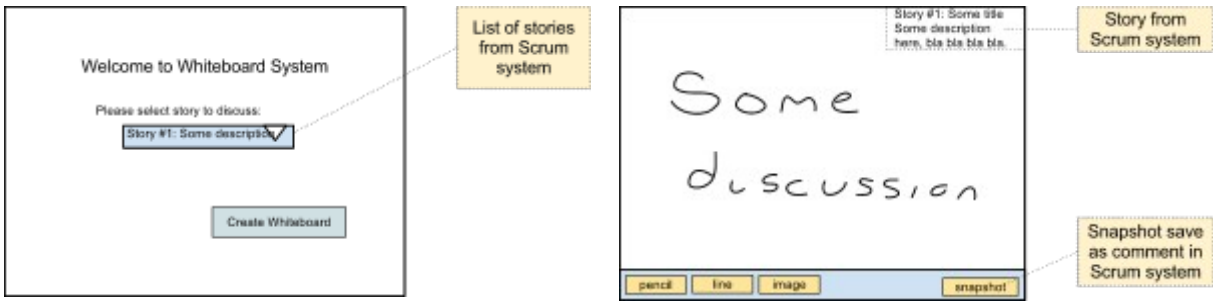
There should be standard drawing tools available on the whiteboard. The tools are as follows:

1. Pencil drawing
2. Line drawing
3. Rectangle drawing
4. Text insertion
5. Image insertion
6. Color selection
7. Eraser.

Select Scrum story to discuss

When planning a discussion about a story in the Scrum system, a whiteboard needs to be created in relation to the Scrum story. A list of stories from the Scrum system should be presented to the user, and possibly shown inside the whiteboard later on during discussion.

This story requires integration with the scrum system (here ScrumDo), in that list of available stories should be read. Figure 5 shows examples of how it could look.


Welcome to Whiteboard System
Please select story to discuss:
Story #1: Some description
Create Whiteboard
List of stories from Scrum system
Some discussion
Story from Scrum system
Snapshot save as comment in Scrum system

(a) When creating a whiteboard, list of stories are presented to the user.

(b) Inside the whiteboard, information about the story being discussed is shown.

Figure 5: Integration of Scrum system stories into whiteboard.

Save snapshots to Scrum system

It should be possible to take a snapshot of a whiteboard and have the snapshot saved as a comment on the related story in the Scrum system. This is useful during discussion, possibly the users may discuss several topics on the same whiteboard, and it can be useful to save an intermediate drawing for later reviewing. Further, it is important that the drawing is available in the Scrum system, so the information is readily available when the users need to continue work on the story at a later time. Figure 5(b) shows an example of a button to save a snapshot in the whiteboard.

Add GUI elements to whiteboard

It should be possible to add a screenshot of an application to the whiteboard, and modify this screenshot by adding new GUI elements on top of it. GUI elements could be buttons, checkboxes, radiobuttons, textboxes, and so on. Figure 6 shows an example of this.

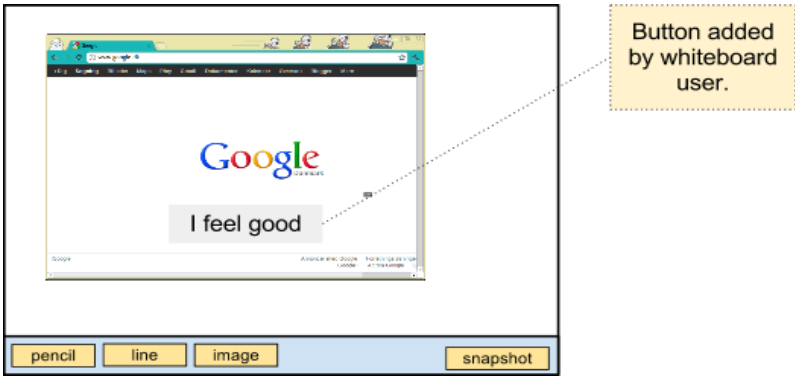


Figure 6: Example of modification of a GUI. A screenshot of Google was added to the whiteboard and the button was modified on it.

Implementation

When the whiteboard application features have now been designed as user stories, we can start with the implementation. Since there is a lot of freedom in this project I do not need much communication with KOKILA during this process, I can make all decisions by myself. In practice however, I have taken KOKILA help to suggest and evaluate many times, mostly because of my insecurity as developer.

In my application, I need to focus on the following:

1. Integration with ScrumDo -- I want to extract the information about the Scrum story which is currently discussed, as well as be able to post outcome of the whiteboard to the story.

2. Web-application drawing -- I do not know about how it is possible to create drawing in web applications. I need to focus on how I can model drawings on the canvas and how they can be sent between clients.
3. Distributed web-application communication -- I need to learn how several browsers can share the whiteboard by synchronizing their state through a server. I will focus on Java on the server-side and Javascript on the client-side.
4. Modeling and saving whiteboard informaiton in database – I need to find how to save this type of information most easily.
5. Snapshot of whiteboard -- I need to learn how a snapshot can be made of a whiteboard and saved on the server-side.

Technology selection

Before starting the implementation, a set of technologies must be selected, which to implement on. In this section I will present my choices, discuss advantages/disadvantages and comment my search for useful libraries given the technologies. Technologies need to be selected for the following tasks:

1. Browser-side application development -- the visual whiteboard in the browser.
2. Server-side application development -- facilitate communication between browser-side whiteboards.
3. Database application development -- facilitate storage of whiteboard data.

In each case I will review available choices and make a choice based on popularity (i.e. how easy is it to find help/examples if I get stuck), availability of libraries (how likely is it that I can find something useful), ease of use (how much time to spend on configuring the IDE/build), own experience and KOKILA's experience.

First, for browser-side application development, there are not so many choices, as mainly Javascript or Flash are the technologies used. When I consider the popularity, Javascript is most usable almost three times more easy use than Flash, according to Google Trends (see Appendix A.1). There appears to be many libraries available for Javascript, while the Flash libraries are more pay-for-use. When considering ease of use, Javascript is also a clear winner: Flash requires compilation before upload, while Javascript is only to edit the file and "reload" the browser window. On the negative side, my own experience with Javascript is limited, however experience with Flash is even less. Based on this, *Javascript* is chosen for browser-side application development.

Secondly, for server-side application development, there are almost too many choices. By examining a timeline of server-side development[1], I can see that the main groups are: Java, Python (Django), Ruby (Ruby on Rails), C# (ASP.NET), PHP and Javascript (Node.js). In the Google Trends popularity measure (see Appendix A.2), Java is clearly most used, followed by PHP and Ruby, Python and C# sharing less popularity. For almost all technologies, there are lots of available libraries. It appears Java has the most standardized libraries, and C# has the most pay-for-use libraries, which is natural since it is the only "closed source" platform. The "scripting" based technologies (Javascript, PHP, Python, Ruby) are more easy since no compilation is required. From my previous experience, I know some Java and some Python,

[1] http://en.wikipedia.org/wiki/Web_development#Timeline Wikipedia, Web development.

and since I have chosen Javascript for the client-side I will learn some Javascript also. I give more weight to my experience and availability of free libraries in Java, I can not get libraries from C# for free for integration and It is extra work to build program for that.

Lastly, for database application development, I imagine stored items to be “document”-type, rather than relational data, hence I would like to use a document-based database technology. Based on my research I have found the following document-based, open source, databases: MongoDB, CouchDB, Redis, FleetDB and OrientDB. I have made a Google Trends popularity measure as shown in Appendix A.3, where MongoDB appears the most popular. FleetDB and OrientDB both score 0. For availability of libraries, I am interested in libraries for the chosen server-side technology (Java). It shows that all except CouchDB have dedicated Java client-libraries. Redis appears not to support queries very well and focus more on managing events. Both my own and KOKILA’s experience with these databases is close to none, however KOKILA has used MongoDB and CouchDB experimentally. Based on this, I chose *MongoDB*.

After selecting technologies, I have searched for useful libraries, with focus on the presented design goals/requirements. The use of libraries can minimize my work, and give me experience in understanding libraries produced from other by reading documentation and watching their examples. I have found (with input from KOKILA developers) the following to be useful, and which to base my project on:

1. Raphaël -- a Javascript library for SVG graphics. This is useful for drawing the whiteboard in browser-window.
2. JQuery -- a Javascript library for easy work with DOM[2] in the browser.
3. JQueryUI -- a Javascript library which can build GUI elements.
4. WebbitServer -- a Java web-server with support for two-way communication to client. This is useful for sending messages between clients through server without polling[3].
5. MongoDB-Driver -- a Java library for communicating with MongoDB from Java.
6. Jackson-mongo-mapper -- a Java library with ORM functions for MongoDB.

Implementation tools and setup

For the programming IDE, I have been using IntelliJ[4]. This is mostly because it is used in KOKILA, and I am able to ask them about help. Also according to KOKILA developers, it is the best tool for Java + Javascript development. IntelliJ supports Apache Maven[5], which is a project management tool for Java projects. Maven allows me to specify dependencies through the IDE and have them automatically downloaded from a centralized repository. Also using Maven, I can manage the lifecycles of the project, such as run the program or run tests, all through IDE. Central to Maven is a “pom.xml” file which configures the project. However, as it is integrated in IntelliJ, there is no need for editing XML, it can be done through the IDE.

KOKILA is using a central code repository on a service called Bitbucket[6], I will use the same place to store the code from the project.

---

[2] DOM is the Document Object Model. A representation of the elements on the HTML page.

[3] Here, polling means to ask server for updates, instead server can send updates to client real-time.

[4] IntelliJ -- http://www.jetbrains.com/idea/

[5] Apache Maven -- http://maven.apache.org/

[6] Bitbucket -- https://bitbucket.org/kereena/whiteboard

Application architecture

When there are several servers and clients at play in a complex system, it can be useful to start with an overview of the application's architecture. This architecture is by no means technical, but shows what the components of the application are and how they talk together.

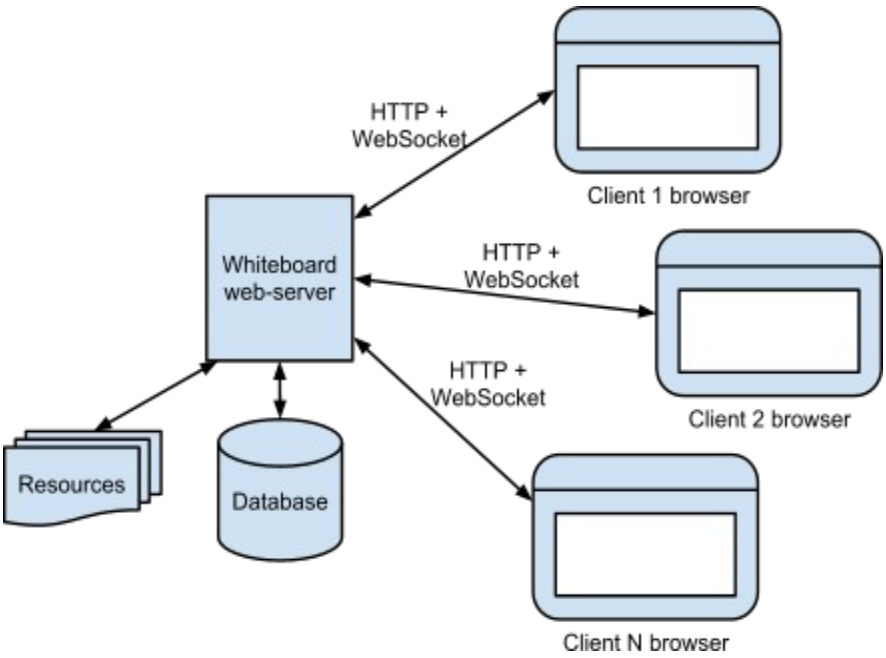


Figure :7 Architecture overview of components in application.

Figure 7 shows the Whiteboard web-server which talks by HTTP and WebSocket protocols to several clients. The Whiteboard web-server also talks to a database for storing and loading the whiteboard data and it uses "Resources" as files for image files.

Protocols

The protocol defines how clients and servers must communicate together. There are three protocols in my system: 1) the URL based protocol for managing whiteboards and support functions; 2) the web-socket based protocol to facilitate communication between server and clients; 3) the client's URL scheme.

JSON encoding

But, first a few words on JSON encoding, since it is used for most data-representation in this project. JSON means JavaScript Object Notation and is a light-weight alternative to e.g. XML encoding. JSON is limited to 6 basic types: Numbers (double precision), strings (double-quoted), booleans (true or false), arrays, objects and null. Especially the arrays and objects are useful. An array is a sequence of elements while objects are key-value pairs. Both can contain nested elements. The table below shows examples of JSON array and JSON object.

[0, 1, 2, “a”, “b”]	{ “a”: 0, “b”: true, “c”: [1, 2] }
(a) Example of JSON array	(b) Example of JSON object

URL based protocol

The URL based protocol can be described by the URLs it defines. The table below list the URLs and their function. Most URLs are accessed by HTTP-GET method. All URLs returns data using the JSON format, which can be processed easily by the Javascript client (browser).

URL	**URL Parameters**	**Description**
/whiteboard/create	t -- title of whiteboard o -- owner. d -- description	Creates a new whiteboard in the system.
/whiteboard/list	(none)	List the whiteboards in the system.
/whiteboard/export	t -- id of whiteboard	Export all information contained in a whiteboard. This is useful for other who want to use the data in other ways.
/resource/upload	t -- id of whiteboard (http headers specifies the file metadata and content the file)	Adds a resource to the system under a given whiteboard. Supported resources are PNG and JPG files. Note this requires HTTP-POST.
/resource/download	r -- id of resource	Downloads a resource.
/scrum/stories	(none)	Retrieves the stories from the Scrum system.

As an example of the URL based protocol, a request and response are shown below. In this case, we create a whiteboard in the system. Note that this is also done by the GUI, but, the URLs can of course be called directly as well. We specify the parameters and get the following URL:
http://localhost:8080/whiteboard/create?t=Discussion+board&o=Kereena&d=None

The system replies with the JSON encoded response message:

```
{"success":true,"boardID":"discussion-board","message":"Created board 'Discussion board'"}
```

Here we can see the ID of the newly created board, and a response message and a success status. Likewise, the other URLs returns their responses in a similar way.

WebSocket based protocol

The WebSocket based protocol is the protocol for real-time updates on the whiteboard between server and clients. Each whiteboard client holds one WebSocket connection to the server. A WebSocket connection is like a TCP connection between a web-server and a browser-client (see my references 3 and 4). The WebSocket connection is initiated from the browser, and will stay open while user is watching the whiteboard. Most web-based applications are pull-based request-response applications, meaning browser requests to "pull" a web-page, and server sends the response. Using WebSockets, the server can push updates to the browser-client without the client needing to pull for them. WebSockets are not (yet) supported in all browsers (only Chrome from what I found).

The whiteboard uses WebSockets to send and receive information from server in real-time. Below is a table of what can be sent over the protocol.

Direction	**Message**	**Description**
C → S	draw,[drawingItemID,drawingItem]	Tells the server that the client has drawn an item on the whiteboard.
C → S	remove,[drawingItemID]	Tells the server that the client has removed an item from the whiteboard.
C → S	move,[x,y]	Tells the server that the client's mouse is at location x, y.
S → C	onjoin,[onlineUser]	Tells the client that another new client has joined the whiteboard.
S → C	onleave,[onlineUser]	Tells the client that another client has left the whiteboard.
S → C	ondraw, [user,drawingItemID,drawingItem]	Tells the client that another user has drawn an item on the whiteboard (the client should update his whiteboard with the item).
S → C	onremove,[drawingItemID]	Tells the client that another client has removed an item from the whiteboard (the client should update his whiteboard by also removing the item).
S → C	onmove,[user,x,y]	Tells the client the position of another client's mouse over the whiteboard.

In the protocol, the first item (the command before the “,”) decides the operation and is a plain text string. The second part is encoded using JSON encoding. As an example seen below, the client draws a text string “Sample whiteboard” on the whiteboard and sends information about it over the WebSocket.

```
draw,{"elementID":"v1338449960953","elementData":
{"type":"text","x":969,"y":157,"text-anchor":"middle","text":"Sample
whiteboard","font":"10px \"Arial\"","stroke":"none","fill":"yellow"}}
```

As can maybe be guessed, there is a relation between the commands starting with “on” and those without. As shown in Figure 8, the client sends “draw” to server, and server sends an enriched (adds username) “ondraw” to each of the clients.

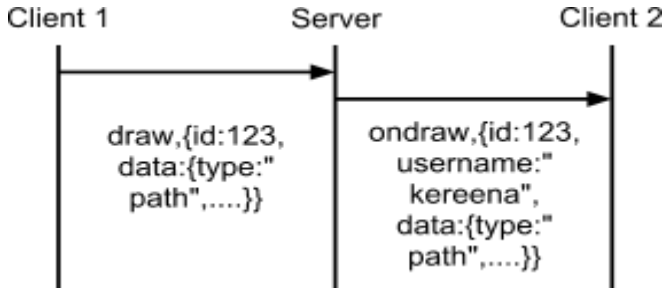


Figure 8: A sample sequence diagram for communication.

The “ondraw” is used also when a new user joins the whiteboard to send all the items in the whiteboard from server to client.

Client GUI protocol

The client is a Javascript application and uses the hash-tag in the URL to decide what state it should be in. There are currently two states supported, as described in the table below.

URL	Description
http://localhost:8080/	Show the welcome page
http://localhost:8080/#!whiteboardID	Show the whiteboard for id = whiteboardID

Drawing in browser

An essential part of the application is the browser-client (the GUI) where actual drawing takes place. As it is not obvious how this can be done and how it can be modeled, I will write about it here.

As previously mentioned the library Raphaël[7] was selected, which is an object oriented Javascript library to create SVG (Scalable Vector Graphics) drawings in the browser. Most browsers supports SVG, and also Chrome.

SVG is an XML format for graphics in the browser. Each graphical element can be represented by some XML. For example a rectangle (with certain filling and border colors) can be represented as follows:

```
<rect style="fill-opacity: 0.2; "
      x="200" y="100" width="100" height="60"
      r="0" rx="0" ry="0" fill="#000000" stroke="#000000"
      stroke-width="2" fill-opacity="0.2" id="v1338584038039">
</rect>
```

Raphaël provides an easy way to refer to and work with the SVG elements. As an example, the above rectangle is created as follows (self.paper is the reference to the Raphaël object):

```
var box = self.paper.rect(200, 100, 100, 60)
      .attr({stroke:'#000','stroke-width':2,
            fill:'#000','fill-opacity':0.2});
```

The "box" variable is a Javascript object, which can be manipulated using Javascript and without knowledge about the underlaying SVG element.

Encoding for sending to server-side

When the item has been drawn in the browser, it must be sent to the server using the protocol ("draw" command). This means that the item must be converted into a string which can be sent. It may be tempting to send the SVG XML element directly, but, Raphaël does not support importing SVG XML elements, so it would be a problem on clients receiving the item. Instead, Raphaël has a function to add an item to the drawing based on the attributes encoded as JSON. This is very good since we are using JSON encoding in the WebSocket protocol. The code below shows how to encode the Raphaël element to JSON by specifying its type and attributes in a Javascript object.

```
// Convert raphael element to Javascript object.
Tools.ToJSON = function(element) {
    var res = {'type': element.type};
    $.extend(res, element.attrs);
    return res;
}
```

On the receiving client, we add the Javascript Object to the Raphaël paper simply by telling Raphaël to create an element and set its attributes, as shown below. (self.paper is the reference to the Raphaël paper)

```
// create element from json
var element = self.paper[json.type]().attr(json);
```

[7] Raphaël Javascript Library -- http://raphaeljs.com/

Serializing and reading the Javascript object to a JSON string is done by JSON.stringify and JSON.parse which are part of the browser's API.

Client side application

As already mentioned the client side application is using the Raphaël library for the parts which are related to drawing in the browser, and for encoding the drawn elements into somthing which can be transmitted between clients (via the server). However, drawing is only one part of the client side application. Here I will try to explain about the structure of the client side and my thoughts on why it is like that. Figure 9 gives an overview of the class structure in the Javascript application.

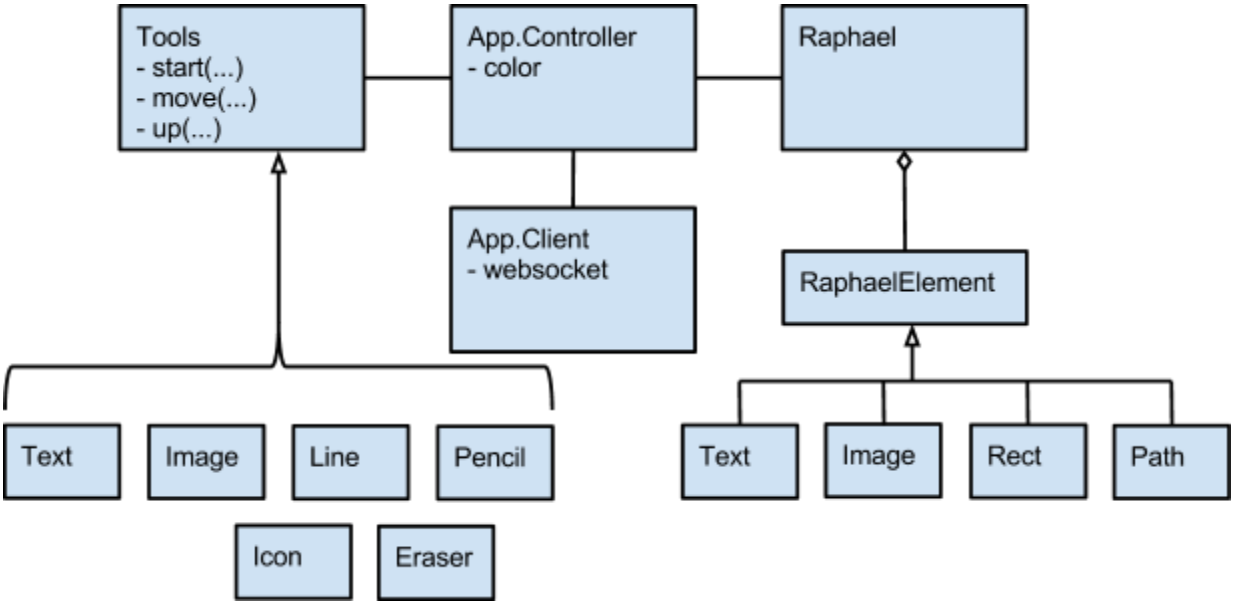


Figure 9: Class diagram for client side Javascript application. Tools have corresponding Raphael elements which they create. Icon and Image both create Image elements and Pencil and Line both create Path elements.

Tools are a central part of the application, as they are responsible for actually drawing something. The App.Controller object listens to mouse movement and mouse clicks on the "canvas" (a HTML DIV element in which the SVG is contained). When the mouse is clicked the tool is activated using its "start()" method, when the mouse is moved while down, its "move()" method is called and finally when the mouse is released again, its "up()" method is called. The tools takes these actions and produces graphical elements through Raphaël from the coordinates. Some tools such as the text and image tools require additional input (i.e. the string representing the text, or the URL to the image), and in that case a dialog box will appear and ask the user for the input before the tool is activated.

A part from the tools are three "states" of the application: To create a new whiteboard from a Scrum story, to join an existing whiteboard, and finally to draw on the whiteboard. The drawing state depends on the existence of a hash-tag in the URL (as shown in protocols section). The join state is the default when no whiteboard is specified, and the create state is currently not accessible directly. To switch between states and to interact with the HTML parts of the web-

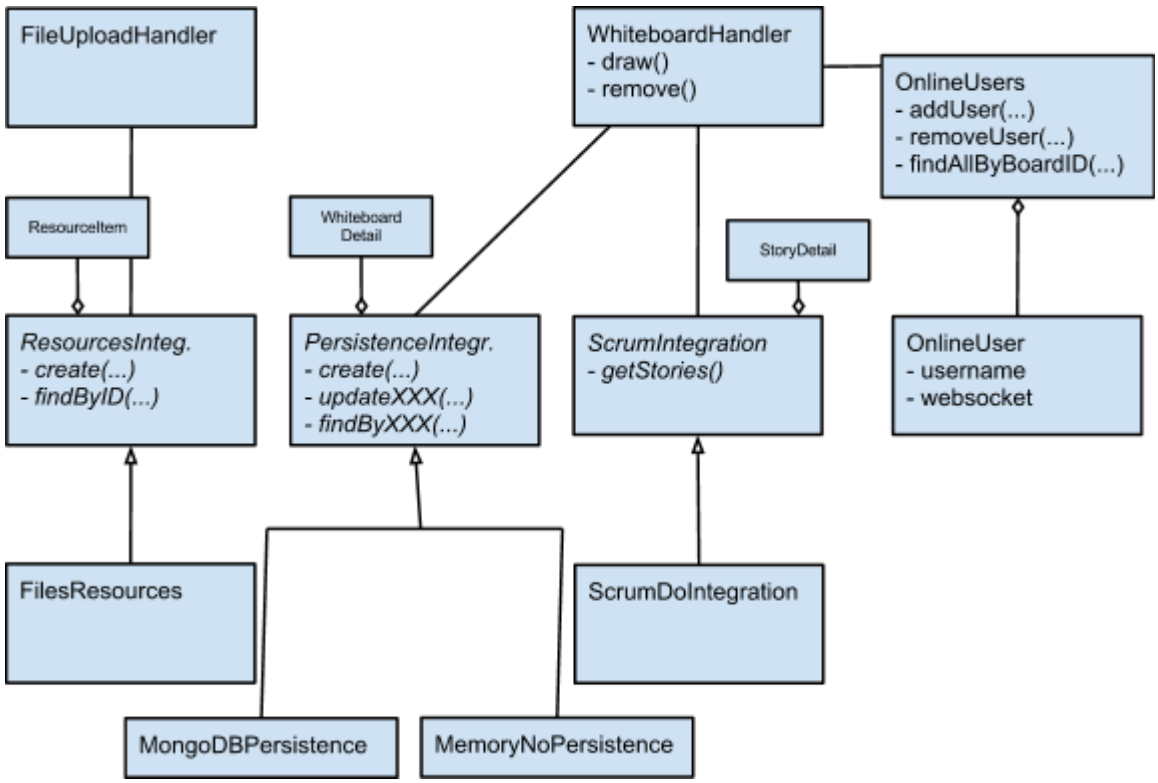
page is all done using JQuery. As an example of assigning a “click” function to the “Join whiteboard” button is seen below:

```
$('#joinButton').click(function() {
        var id = $('#joinInput').val();
        window.location.hash = '!' + id;
        activateWhiteboard($('#username').text(), encodeURI(id));
});
```

The $(‘#joinButton) refers to an element in the HTML page with the “id” attribute set to “joinButton”. This button’s click handler is set to the given function.

Server side application

The server side of the application is built on top of WebbitServer[8], which is a Java web-server that can handle WebSocket connections and normal HTTP requests. In the WebbitServer, one HttpHandler corresponds to one URL path. For example “FileUploadHandler” class is bound to the URL “/resource/upload”. All the binding takes place in the Main method, which is the main entry point for creating the server. The WebbitServer is not part of any large toolkit, so in that way it is very simple to setup and get started with. Figure 10 shows the class diagram of main classes in the server-side of my application.



[8] WebbitServer -- http://webbitserver.org/

Figure 10: Server-side class diagram for selected web handlers: 1) WhiteboardHandler, which is the handler for WebSocket connections; 2) FileUploadHandler which handles the uploaded image files.

Database design

As mentioned, the selected MongoDB is a document based database, a popular name for it is “NoSQL” database. MongoDB is very different from the relational databases I know.

Firstly, it does not require any schemas to be defined. It has the concept of “collections”, which can be thought of as tables, where any document can be inserted. In MongoDB, a document is specified using BSON, which is an extension of JSON, allowing for binary data. Since it is an extension, it means that normal JSON objects can be saved in the database. So, a “collection” in the databse is a collection of JSON objects.

Although I am not required to specify tables, it is of course still useful for myself to specify the expected structure of a document in the database. A document has to model what a whiteboard looks like. Figure 11 shows the structure of one entry in the database. “(“ and “)” are used to specify a list of items and “{“ and “}” are used to specify an object (meaning name/value pairs).

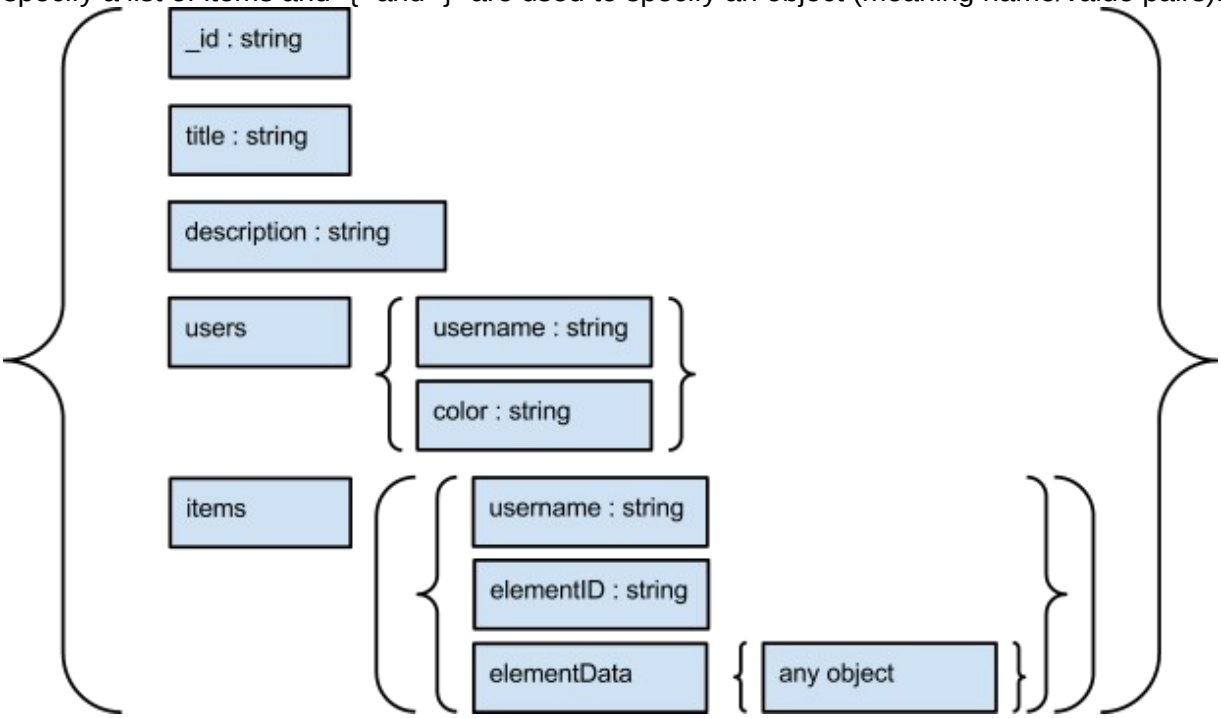


Figure 11: Database design for collection “whiteboards”.

Below is an example of a whiteboard document in the database, with a single user and two drawn items.

```
{ "_id" : "discussion-board",
  "title" : "Discussion board",
  "description" : "Discuss about Story 1",
  "users" : { "Kereena": "blue" },
```

```
  "items" : [
    { "username" : "Kereena",
      "elementID" : "v1338449803168",
      "elementData" : { "type" : "image", "x" : 201,  "y" : 54,
                        "width": 629,     "height": 437,
                        "src": "resource/download?r=123/sample.jpg" } },
    { "username" : "Kereena",
      "elementID" : "v1338449833417",
      "elementData" : { "type": "rect", "x": 469, "y" : 307,
                        "width": 213, "height" : 35,
                        "r" : 0, "rx" : 0, "ry" : 0,
                        "fill" : "white", "stroke" : "white",
                        "stroke-width" : 2, "fill-opacity" : 0.2 }
    }
  ]
}
```

When inserting values into the database the JSON format does not need to be specified, since the Jackson-mongo-mapper[9] library was used. This library converts from simple Java value objects into JSON objects, which can be inserted into the database. It is a very simple form of an ORM. The library wraps a MongoDB collection as a typed collection by specifying type of elements in the collection and the type of the key. The wrapping code can been in the code extract below:

```
DBCollection mongoCollection = database.getCollection("whiteboards");
        this.whiteboards = JacksonDBCollection.
            wrap(mongoCollection, WhiteboardDetail.class, String.class);
```

Once the collection is wrapped, it is naturally expected that all elements in the collection must conform to the typed wrapping. This means that the ORM is convenient, but, it restricts the dynamic nature of the schema-less database. Below is an example of inserting a Java object into the database, “WhiteboardDetail” is a simple Java value object:

```
// create object to save
WhiteboardDetail detail = new WhiteboardDetail();
detail.description = description;
detail.owner = owner;
detail.title = title;
detail.boardID = Slugify.slugify(title);
// save it
WriteResult result = this.whiteboards.insert(detail);
```

Once a document has been added to the database it can be queried and found. In the whiteboard application the only querying is done by ID. As the wrapped collection is already typed this operation is very simple:

```
public WhiteboardDetail findByBoardID(String boardID) {
      return whiteboards.findOneById(boardID);
}
```

[9] Mongo-jackson-mapper -- https://github.com/vznet/mongo-jackson-mapper

Updating documents in MongoDB is done very similar to SQL UPDATE commands. Each operation on the database is transactional, but, it is not possible to combine several operations into one transaction. In MSSQL for example, it is common to SELECT and then UPDATE within the same transaction. There are several additional update commands according to SQL however, because of document structure. In the whiteboard application three update commands are used:

1. set -- corresponds to the traditional set in SQL, allows to set the value of a field, however here also nested field values can be set.
2. push -- will add a new element to a field of type “array”.
3. pull -- will remove an element from a field of type “array”.

As a code example the push operation is shown below, where a new “DrawingItem” is added to the array of items in the whiteboard:

```
public WhiteboardDetail addDrawingItem(String boardID,
                       WhiteboardDetail.DrawingItem drawingItem) {
  WhiteboardDetail detail = whiteboards.findOneById(boardID);
  detail.items.add(drawingItem);
  WriteResult<WhiteboardDetail, String> result =
    whiteboards.updateById(boardID, DBUpdate.push("items",drawingItem));
  return detail;
}
```

Only few features from the MongoDB were used for the whiteboard. Apart from the schema-less structuring, some of the advanced features are automatic failover/replication and map/reduce operations, which are each a huge topic by themselves.

Discussion

Product evaluation

Evaluating the result of the development is always important. As this is only a prototype, we can expect many feedback, which can be negative or about improvements, but hopefully for my project I can get indicator if the developed system is in the right direction or all wrong. A comprehensive evaluation is of course best, but due to time limits of my project, I have only managed to get two sessions with end-users from KOKILA. In both sessions a task related to HTML pages were discussed. The output from both sessions are found in Appendix C.

The input I wanted to get from the system was:

1. What was easy using the whiteboard?
2. What was hard or seemed strange using the whiteboard?
3. Any ideas for what could make it more useful?
4. Did the whiteboard help with your task?

For the first question, they all agree that the developed virtual whiteboard was easy to get started with and it was good to visualize what they were talking about. Special focus was on that

they said that features of including previous work and “extending” it on the whiteboard was good. Also developer input was that they liked the ability to give feedback instead of one-way communication.

For the second and third questions, a few technical issues were brought up. This was quite as expected as it is a prototype. Generally the tools should function like in “normal” drawing programs, here especially the “text” tool was not standard, and other standard functionality such as “move” and “undo” is not yet supported. New features suggested and not on the story-list were: Support for touch-devices and better “screen” management.

The last question is the most important for me, as it evaluates the usefulness of the whiteboard in relation to outsourced scrum problems. Luckily all are generally positive. However, given input from other questions, I can see that still some features are missing before it is truly useful. That said, the whiteboards produced during the evaluations sessions are already impressive to me, and it appears they found it useful for the tasks at hand.

Application of Pba course knowledge

In this section I relate the courses from Pba. Software in this project. For each course I will discuss what has been done, what thoughts I have done on the topic and what future directions I could take.

Test

Testing resolves with the task of showing the presence of errors in the software. In the course several techniques for testing were presented, all involving modeling software under test, the formulation of testing requirements (describe what to test) and test cases (actual tests which “covers” the requirements). Here I perform testing of only a single component of the project code, under the assumption that if I can test one part, probably I can test it all. I can see that testing Java server code is much easier than the Javascript GUI code, but, it should be only a question of time and effort.

In testing one of the models is “graph based”, which means that the software under test is modeled using a graph. To test the model requires test requirements, which means how to cover the graph. Examples of such coverages are: Node coverage (all nodes are covered), edge coverage (all edges are covered), node-pair coverage (all node-pairs are covered) or prime path coverage where all prime paths are covered.

Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	21	41%	112/272	25%	19/74	1.883
slugify	1	88%	8/9	50%	4/8	4
whiteboard.colors	2	100%	3/3	N/A	N/A	1
whiteboard.persistence	3	100%	15/15	N/A	N/A	1
whiteboard.persistence.memory	1	83%	31/37	66%	8/12	3.167
whiteboard.persistence.mongodb	1	89%	25/28	50%	1/2	1.833
whiteboard.stories	2	100%	8/8	N/A	N/A	1
whiteboard.stories.scrumdo	1	100%	22/22	75%	6/8	2.333
whiteboard.web	9	0%	0/144	0%	0/40	1.885
whiteboard.web.util	1	0%	0/6	0%	0/4	3

Report generated by Cobertura 1.9.4.1 on 5/30/12 9:55 AM.

Figure 12: Cobertura coverage report overview. N/A means there are no branches in the given code (for example interfaces or data objects).

To make use of available tools, I have selected Cobertura[10], which is a graph model based coverage measurement tool. It works as a plugin to my build procedure, and produces output in form of HTML pages, where I can see for each class and each method and each line what percentage is covered by test cases. Cobertura is, as said, graph based coverage but supports only node and edge coverage (called “branches”). According to testing course, these are the simplest forms of coverage. If I want to make more advanced coverage, e.g. for prime paths, I must evaluate and calculate coverage by hand, so that is why I choose Cobertura. Figure 12 and 13 shows examples of Cobertura output. Generally, Cobertura seems a very easy tool to use and useful for testing.

```
58        @Override
59        public WhiteboardDetail addUser(String boardID, String username) {
60  1         WhiteboardDetail detail = whiteboards.findOneById(boardID);
61  1         if (detail.users.containsKey(username))
62  0             return detail;
63  1         detail.users.put(username, colors.getColor(username, detail.users.size()));
64  1         WriteResult<WhiteboardDetail, String> result =
65                  whiteboards.updateById(boardID, new DBUpdate.Builder().set("users." + username, detai
66  1         return detail;
67        }
```

Figure 13: Cobertura report. Example of a branch which is highlighted as not reached by my test cases.

Databases for developers

Databases handles persistence of data, meaning data is available even though the web-application is restarted and its memory reset. In the course, several topics of databases were discussed, many of which were related to the use of SQL. Since my selected database, MongoDB, however is not an SQL database (they call it a NoSQL database), the SQL related techniques were not used, and I had to find other techniques.

[10] Cobertura -- http://cobertura.sourceforge.net/

Comparing to MSSQL server, the most close is the XML attributes, which can also be queried and are semi-structured data. In this way I find that a collection in MongoDB is quite similar to a table with a single XML field in MSSQL. XML is one way to “encode” the content of the document used by MSSQL, but, the graph/tree structure of XML can also be represented as JSON, thus they are structurally the same. As an example seen below:

<whiteboard>
  <id>123</id>
  <title>ttttt</title>
  <users>
  <user>Kereena</user>
  <user>bbbb</user>
  </users>
</whiteboard>
	{
	  id: 123,
	  title: “ttttt”,
	  users: [
	  “Kereena”,
	  “bbbb”
	  ]
	}

XML	JSON

There is an important difference from the Pba course however. XML documents have the feature to be specified by an XML-schema, and the MSSQL server supports to enforce this XML-schema on the XML values entered into the database. MongoDB and its JSON format does not have any system for this, it is completely without any ability to force structure, it is up to the application developer.

In the query language to specify fields, the MSSQL XML format is clearly more advanced by allowing XPath expressions. MongoDB offers only a dotted notation, for example “whiteboard.users”.

As already mentioned in implementation, MongoDB is a mature document-database, and it has many advanced features for e.g. permissions control, backup and restore, distributed databases and map/reduce functions on data. However, these features have not been applicable in my applications and thus I don’t have experience to report with them.

System integration

System integration course was about how to design services for use by others, and how to make a well-structured set of services. In this project, the HTTP-URL based protocols can be defined as web-services, which are then used by the GUI component. Further, my project has been consuming services from the ScrumDo system. Here I will relate these services to the SOA principles from the course, by going through each principle and relating it to how I used it when designing:

Reusability: This principle concerns the case that my services must be usable in several contexts, not only for my use. The “resource” services are clearly reusable, they facilitate something as general as image storage and download. While the resources relates to a whiteboard’s ID, this ID could also be a from another system. The whiteboard services can also be argued to be reusable, since they could allow 3rd party applications to integrate/visualize data from the whiteboards created in my system.

Granularity: This principle concerns with the modularity of the services, i.e. how they are grouped. In this case the services have been grouped by URL into two. Those services concerning with whiteboard operations are all on a URL starting with “/whiteboard” while the resource concerned services are on URLs starting with “/resource”.

Loose coupling: This principle concerns with how services are tied to each other. It is another view of reusability where reusability is limited if services are tightly coupled (i.e. have many dependencies on each other). As already discussed my services are grouped into two functional areas, which are not dependent on each other, thus they are loosely coupled too.

Standardized service contract: This principle concerns with how services are implemented using standards. Since the services in the project are using only standard HTTP communication and standard JSON encoding of answers, they can be said to be highly standardized.

Abstraction: This principle concerns with how services hide internal logic of the server from the clients. In this point the services do not fulfill very well. Especially the “export” function exposes the internal structure of the whiteboard. Since there is only one version of the whiteboard application currently, then it is not so relevant, and a promise of future versions to be backward compatible could help to make this principle better implemented. The other services are quite abstract in that they don’t expose internal logic or structure.

Autonomy: This principle concerns with how well a service can control the environment it depends on. For example, the service “/scrum/stories” which imports stories from the ScrumDo system can be said to have a low degree of autonomy, since it does not control the ScrumDo system, nor the Internet which connects to the ScrumDo system. The other services depends on the MongoDB, which the service does not itself have control over, but which the administrator of the application can control. In this regard there are few dependencies, and it is safe to say that except the ScrumDo integration, then the system has a medium degree of autonomy.

Statelessness: This principle concerns with how little information (state) a service need to function. State is defined here as “user-relevant information in memory”. For example, the whiteboard WebSocket interface need to keep the WebSocket state for each connected user, which also contains the username. Generally however, the system is very stateless. The web-services each process one request at a time without keeping any state, and the WebSocket interface only keep the username as state, everything else is processed (sent to other clients and persisted in database) and then forgotten.

Discoverability: This principle concerns with how services publish descriptions of their functionality, so other (new) clients can use them. On this point nothing has been done in the whiteboard application, except from the documentation written in this report.

Composability: This principle concerns with how easily services can be put together in new uses. As already described, there are two function areas, whiteboard and resources. This allows for a high degree of composability by other applications wishing to integrate such functionality.

Generally I would say that the principles have been applied successfully, mostly on the web-service HTTP URLs as described in the Protocols section. This is in my opinion fine, as the other part of the application is the GUI.

Contract based development

Contract based systems is about proving the correctness of the developed code, and about specifying the contract between client and server, here the client and server means calling code and called code. For example, to ensure that an object is used correctly and to guarantee that the implementation of the object can be depended on. Code contracts have three concepts to specify: Pre conditions (what has to be true before calling method), post conditions (what has to be true after calling method) and class invariants (what has to be true at all times for the object).

In the Contract based development classes, we only used Code Contracts from Microsoft, however this product is not applicable to my Java and Javascript codes. After invetigating, however, I found that there are many “code contract” libraries available for Java, and the library Cofoja[11] was selected for implementation. As this was done at the end of the project however, it was not yet fully implemented. Some few classes had contracts added, using Java annotations. As an example, parts of WhiteboardDetail class is shown below to demonstrate how contracts look using this library:

```
@Invariant({
        "users.size() <= items.size()", // there cant be more users than items
        "boardID != null", // boradID must be there.
        "title != null" // title must be there.
})
public class WhiteboardDetail {
[parts removed here]
   @Requires({
             "item.username != null", // we need username
             "item.elementID != null", // we need id
             "item.elementData != null", // we need data
             "item.elementData.attr('type') != null", // type attribute req.
             "users.containsKey(item.username)" // req username registered.
    })
    @Ensures("items.size() == old(items.size()) + 1")
    public void addDrawingItem(DrawingItem item) {
        items.add(item);
    }
[parts removed here]
}
```

What we can see, is that it is possible to specify code contracts easily using annotations. If a comparison is made with the Code Contracts library from Microsoft, then there are some differencies:

1. in Code Contracts the contracts are code, meaning they are compile-time checked.
2. in Cofoja contracts are comments, meaning no compile-time check.
3. in Code Contracts their implementation is integrated into the IDE.
4. in Cofoja it is required that the java program must be started in a special way by hand.

It is the last part which I did not have time to experiment with, hence I only specified few contracts, but did not have them executed runtime. However, the library is promising and I am sure that given time contracts can be implemented into the project.

Development of large systems

[11] Cofoja from Google -- http://code.google.com/p/cofoja/

Development of large systems is a course which was centered about two things: Firstly, a practical part with how to build products based on consuming other Internet services. Secondly, a theoretical part on how to create systems which can scale. The first part I have already dealt with under System integration, so I will only write about my approach to scalable system here. Scalable means the ability to handle infinite users.

My system does not have any built-in scalable parts except the database MongoDB, however I think that it is still scalable. The thoughts I have done about the scalability of my system are: Firstly, each whiteboard is limited to one server in the system. Secondly, the scenario of which my system is based upon, handles about 3-4 users who will discuss on a whiteboard, not large groups. This means that there will be many "small" whiteboards, and that I can install further servers running my application when the number of small whiteboards get too much. As mentioned above MongoDB is already scalable.

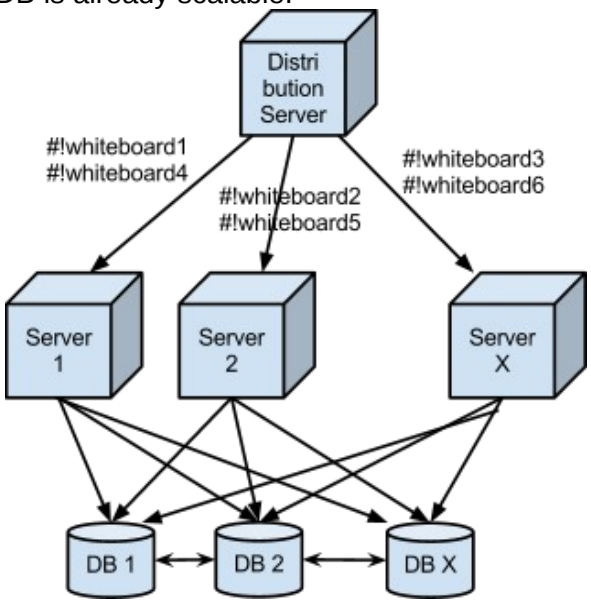


Figure 14: Scalability setup for whiteboard services.

Figure 14 shows an overview of such an architecture. The distribution server is already available, for example NGINX[12], which can assign users to a specific server, based on elements in the URL (in my case the whiteboard ID). Based on this, while my application does not have scalable elements, it can be easy to make it scalable using available technology.

Process evaluation

This part evaluates the process of writing a Pba. thesis, and specifically how it went in this case. First I would like to thank the KOKILA team for giving good inputs to me and for participating in (almost) all the meetings I scheduled. There was quite some confusion in the beginning and it took some time before the project was established and I figured out what to do. Once it was

[12] NGINX -- http://wiki.nginx.org/HttpUpstreamModule

established, following the Scrum process model and using the ScrumDo system helped to coordinate what to do next. Delivering the stories at the agreed deadlines was also a challenge for me. Too often stories were selected but had to be pushed to next iteration. At the end of the project, I still have outstanding stories which were not possible to finish in time.

The help from the strategy analysis on risks, strength/weaknesses and stakeholders helped me to focus on when and who I can ask when there was problems.

All in all, given the short timeframe for the project, the progress is acceptable and a product was produced by the support of the process.

Conclusions

To conclude this report a summary and answers to the stated problem definition can be presented. First, Scrum and problems related outsourcing were referred. This allowed to select a problem solution statement. Secondly, followed the design and implementation of the proposed solution. Finally, discussion was made on the evaluation of the solution and the process, together with application to the Pba courses. What is left, is to compare the presented solution with the problem definition.

What are the major problems areas in agile software development related to outsourcing?
Here I found the major problem area to be communication. The review of the Scrum process indicated that Scrum is a method which requires extensive communication between product owner and developers. The literature review on outsourcing problems confirmed this need for communication, and it was also found to be a problem within KOKILA.

What are the solutions can be proposed to build the system that handles specific problems related to agile with offshore?
Together with KOKILA, a system was proposed: Virtual whiteboard with support discussing GUI elements. The virtual whiteboard addresses the need to build a richer communication channel between developers and product owner, thus (partially) handling the communication problem.

Evaluate what is the user's feedback related to the prototype of the system built?
The system was evaluated using two case scenarios with KOKILA and their customers.The evaluation provided feedback that the proposed whiteboard solution helps to improve communication, but needs to be further matured.

These answers conclude the report: Our solution can help to solve the communication problem in outsourced Scrum, but, further work on the technical parts need to be done before it is truely useful.

Acknowledgements

I would like to say many thanks to the people who helped me during the project: My ScrumMaster Søren for helping with input/mentoring on the Scrum process, KOKILA developers for brainstorming, technical discussion and evaluation sessions and especially product owners Rasmus and Jens for their time. Special thanks to my supervisor Bent for helping with foundations of project, and spending his weekend on my last minute feedback.

References

[1] S. Gopalan and O. Embry, *Outsourcing to India - current and emerging trends: Implications for global managers*, AIB-SE 2003 Annual Meeting.

[2] R. Diana, *Agile and the art of outsourcing*, Dzone, 2011, http://agile.dzone.com/news/agile-and-art-outsourcing

[3] Mathieu Carbou, *Reverse Ajax*, IBM developerWorks, 2011, http://www.ibm.com/developerworks/web/library/wa-reverseajax1/index.html

[4] Eric Freemand, *Getting started with HTML5 and WebSockets,* Wickedly Smart, 2012, http://wickedlysmart.com/2012/getting-started-with-html5s-web-sockets/

[5] Karl Seguin, *The Little MongoDB Book*, OpenMyMind.Net, 2011, http://openmymind.net/2011/3/28/The-Little-MongoDB-Book/

[6] Project Foundation, Bent. H. Pedersen,

[7] Bjarne Kousholt, Project Management Theory and Practice, 1st Ed.

[8] Javascript tutorial, http://www.w3schools.com/js/default.asp

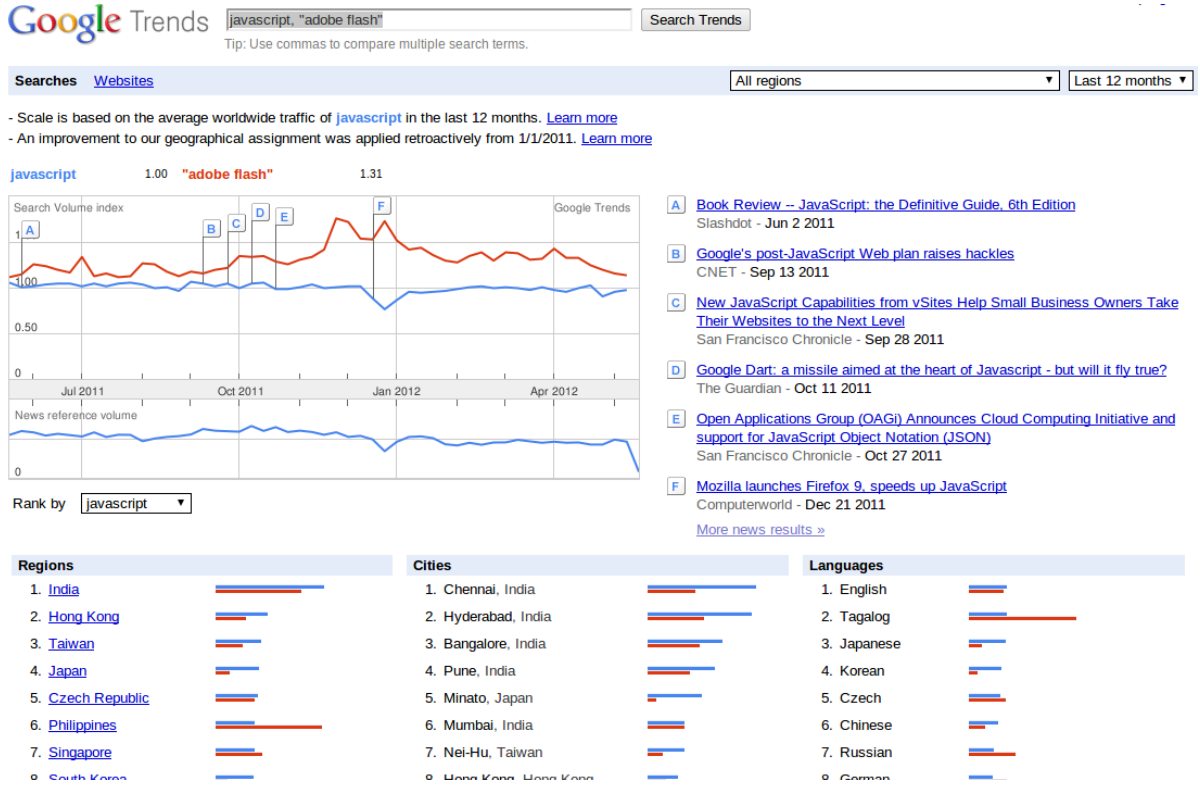
[9] MangoDB tutorial, www.mongodb.org/

Appendix

Appendix A: Google Trends analysis

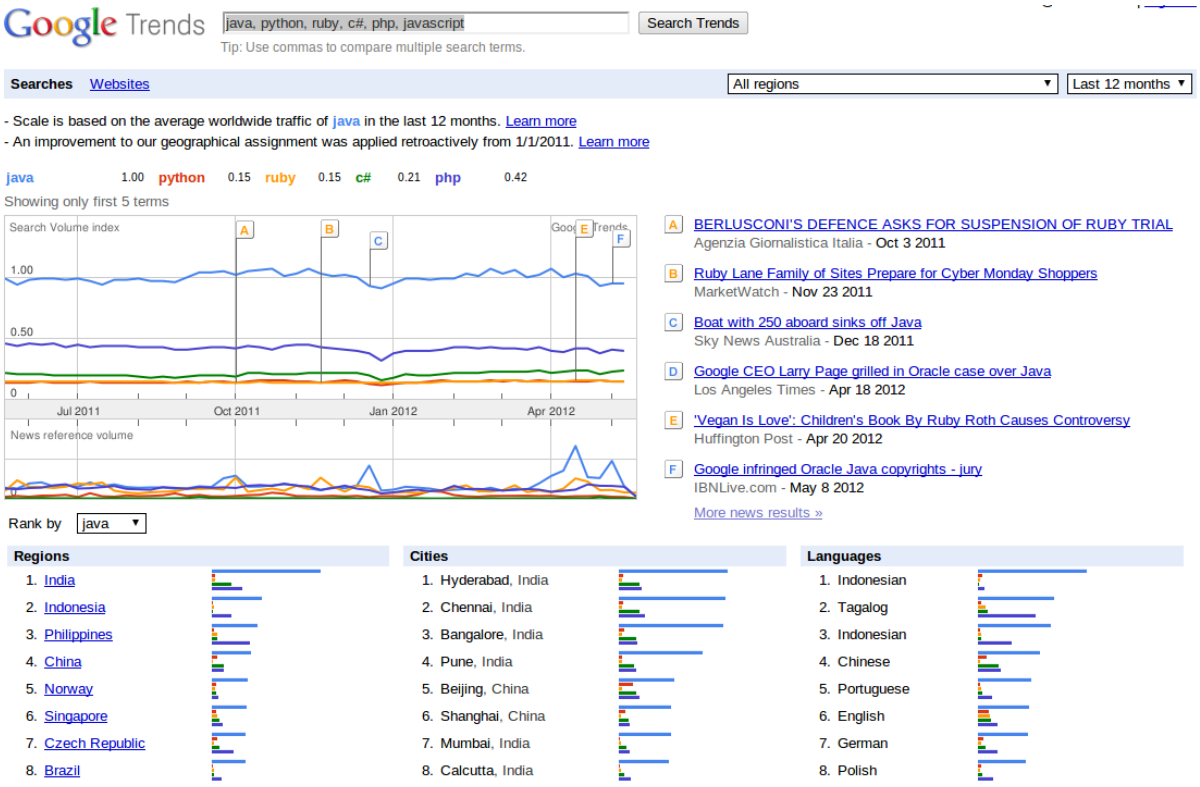
A.1 Client-side technology trends

URL: http://www.google.com/trends/?q=javascript,+"adobe+flash"


Google Trends
javascript, "adobe flash"
Search Trends
Tip: Use commas to compare multiple search terms.
Searches
Websites
All regions
Last 12 months
- Scale is based on the average worldwide traffic of javascript in the last 12 months. Learn more
- An improvement to our geographical assignment was applied retroactively from 1/1/2011. Learn more
javascript 1.00 "adobe flash" 1.31
Search Volume index
Google Trends
1.00
0.50
0
Jul 2011
Oct 2011
Jan 2012
Apr 2012
News reference volume
0
Rank by
javascript
A Book Review -- JavaScript: the Definitive Guide, 6th Edition
Slashdot - Jun 2 2011
B Google's post-JavaScript Web plan raises hackles
CNET - Sep 13 2011
C New JavaScript Capabilities from vSites Help Small Business Owners Take Their Websites to the Next Level
San Francisco Chronicle - Sep 28 2011
D Google Dart: a missile aimed at the heart of Javascript - but will it fly true?
The Guardian - Oct 11 2011
E Open Applications Group (OAGi) Announces Cloud Computing Initiative and support for JavaScript Object Notation (JSON)
San Francisco Chronicle - Oct 27 2011
F Mozilla launches Firefox 9, speeds up JavaScript
Computerworld - Dec 21 2011
More news results »
Regions
1. India
2. Hong Kong
3. Taiwan
4. Japan
5. Czech Republic
6. Philippines
7. Singapore
Cities
1. Chennai, India
2. Hyderabad, India
3. Bangalore, India
4. Pune, India
5. Minato, Japan
6. Mumbai, India
7. Nei-Hu, Taiwan
Languages
1. English
2. Tagalog
3. Japanese
4. Korean
5. Czech
6. Chinese
7. Russian

A.2 Server-side technology trends

URL:
http://www.google.com/trends/?q=java,+python,+ruby,+c%23,+php,+javascript


Google Trends
java, python, ruby, c#, php, javascript
Search Trends
Tip: Use commas to compare multiple search terms.
Searches Websites
All regions
Last 12 months
- Scale is based on the average worldwide traffic of java in the last 12 months. Learn more
- An improvement to our geographical assignment was applied retroactively from 1/1/2011. Learn more
java 1.00 python 0.15 ruby 0.15 c# 0.21 php 0.42
Showing only first 5 terms
Search Volume index
1.00
0.50
0
Jul 2011
Oct 2011
Jan 2012
Apr 2012
News reference volume
A BERLUSCONI'S DEFENCE ASKS FOR SUSPENSION OF RUBY TRIAL
Agenzia Giornalistica Italia - Oct 3 2011
B Ruby Lane Family of Sites Prepare for Cyber Monday Shoppers
MarketWatch - Nov 23 2011
C Boat with 250 aboard sinks off Java
Sky News Australia - Dec 18 2011
D Google CEO Larry Page grilled in Oracle case over Java
Los Angeles Times - Apr 18 2012
E 'Vegan Is Love': Children's Book By Ruby Roth Causes Controversy
Huffington Post - Apr 20 2012
F Google infringed Oracle Java copyrights - jury
IBNLive.com - May 8 2012
More news results »
Rank by java
Regions
1. India
2. Indonesia
3. Philippines
4. China
5. Norway
6. Singapore
7. Czech Republic
8. Brazil
Cities
1. Hyderabad, India
2. Chennai, India
3. Bangalore, India
4. Pune, India
5. Beijing, China
6. Shanghai, China
7. Mumbai, India
8. Calcutta, India
Languages
1. Indonesian
2. Tagalog
3. Indonesian
4. Chinese
5. Portuguese
6. English
7. German
8. Polish

A.3 Database technology trends

URL: http://www.google.dk/trends/?q=couchdb,+fleetdb,+mongodb,+orientdb,+redis

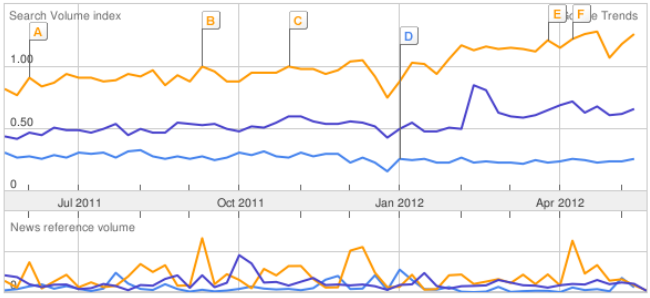
Google Trends couchdb, fleetdb, mongodb, orientdb, redis Search Trends

Tip: Use commas to compare multiple search terms.

Searches Websites All regions Last 12 months

- Scale is based on the average worldwide traffic of mongodb in the last 12 months. Learn more
- An improvement to our geographical assignment was applied retroactively from 1/1/2011. Learn more

couchdb 0.26 fleetdb 0 mongodb 1.00 orientdb 0 redis 0.55



Rank by mongodb

A MongoDB Competes on Speed and Flexibility
PC World - Jun 8 2011

B 10gen, Sponsor of MongoDB, Announces $20M in Funding
MarketWatch - Sep 12 2011

C Viber Media Selects MongoDB as Database in the Cloud for Popular Mobile Apps
MarketWatch - Nov 1 2011

D CouchDB daddy unplugs from Apache hive
Register - Jan 6 2012

E dotCloud Delivers on the Promise of Full-Stack Development With Support for Websockets, MongoDB 2.0 and Vertical Scaling
MarketWatch - Mar 27 2012

F EPAM Big Data Competency Center and 10gen Partner to Bring MongoDB to Mission-Critical Enterprise Applications
MarketWatch - Apr 12 2012

More news results »

Regions

1. South Korea
2. China
3. Hong Kong
4. Russian Federation
5. Ukraine
6. Sweden
7. Taiwan

Cities

1. Beijing, China
2. Hangzhou, China
3. San Jose, CA, USA
4. Shenzhen, China
5. Guangzhou, China
6. Seoul, South Korea
7. Shanghai, China

Languages

1. Korean
2. Chinese
3. Russian
4. Swedish
5. English
6. Finnish
7. Japanese

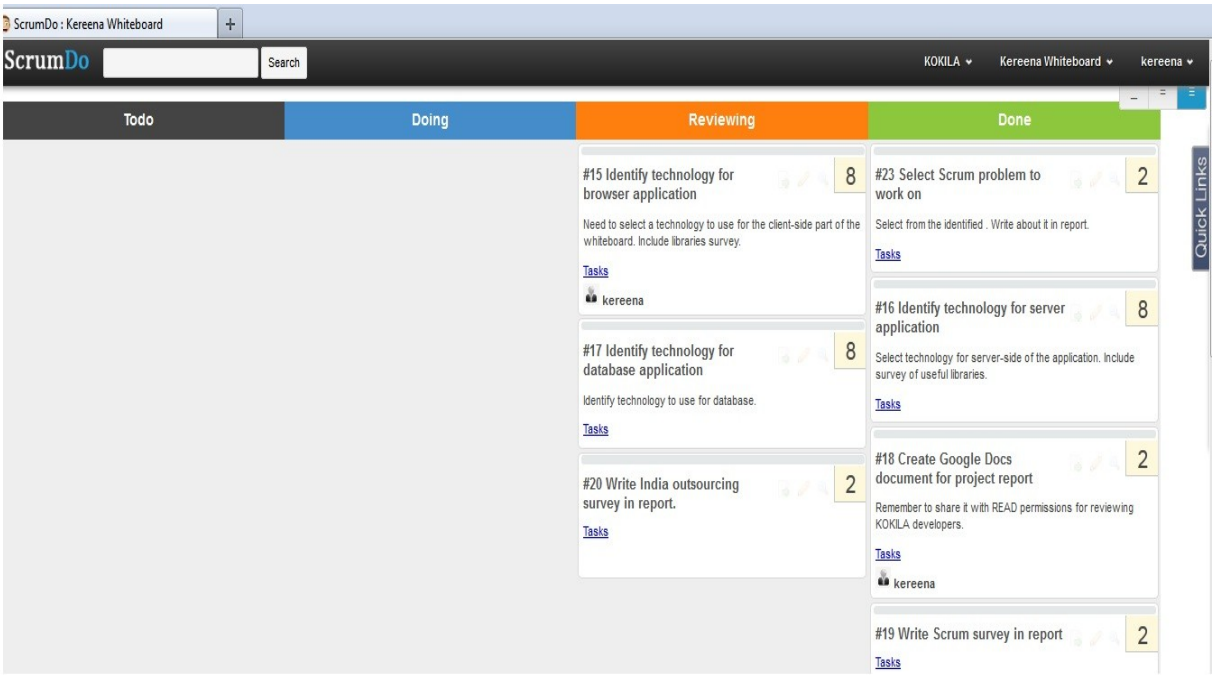
1

Appendix B: ScrumDo documentation

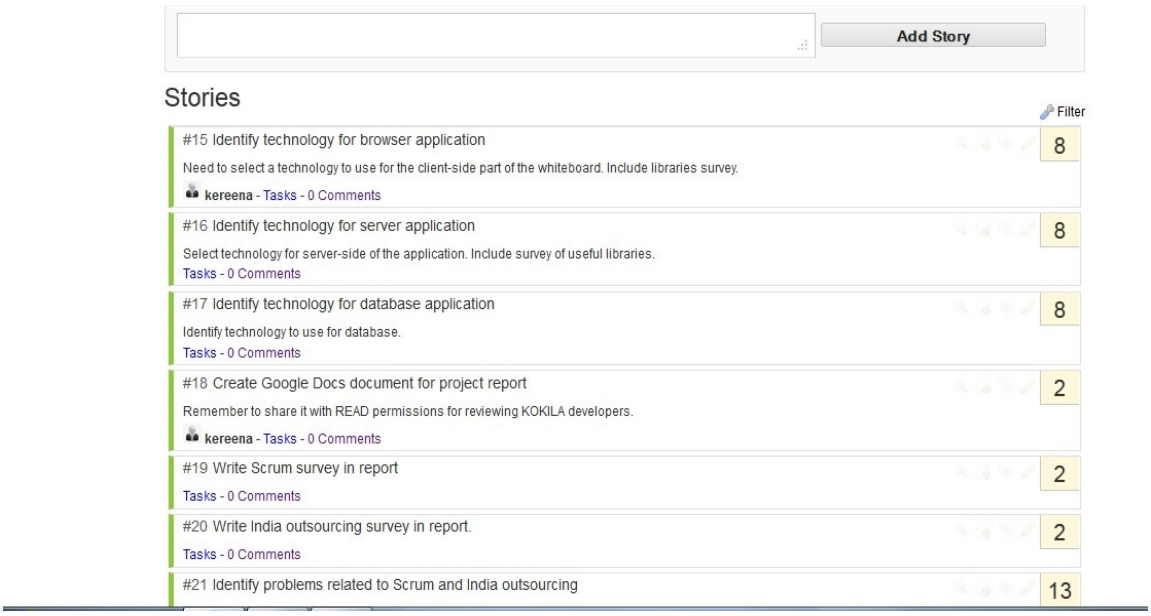
This Appendix contains screenshots from ScrumDo system, showing which stories have been worked on.

B1. Iteration 1

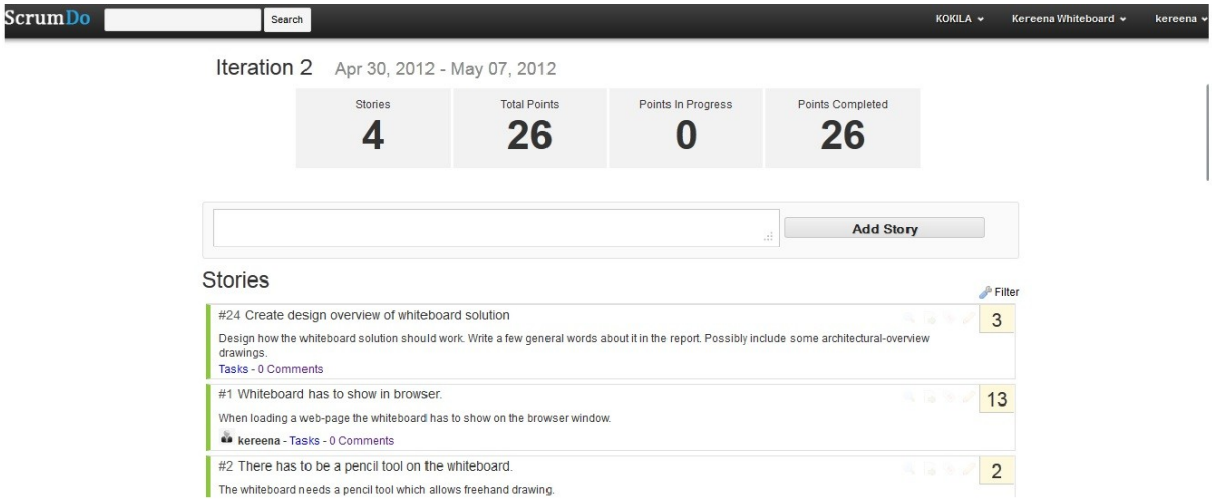
Dashboard



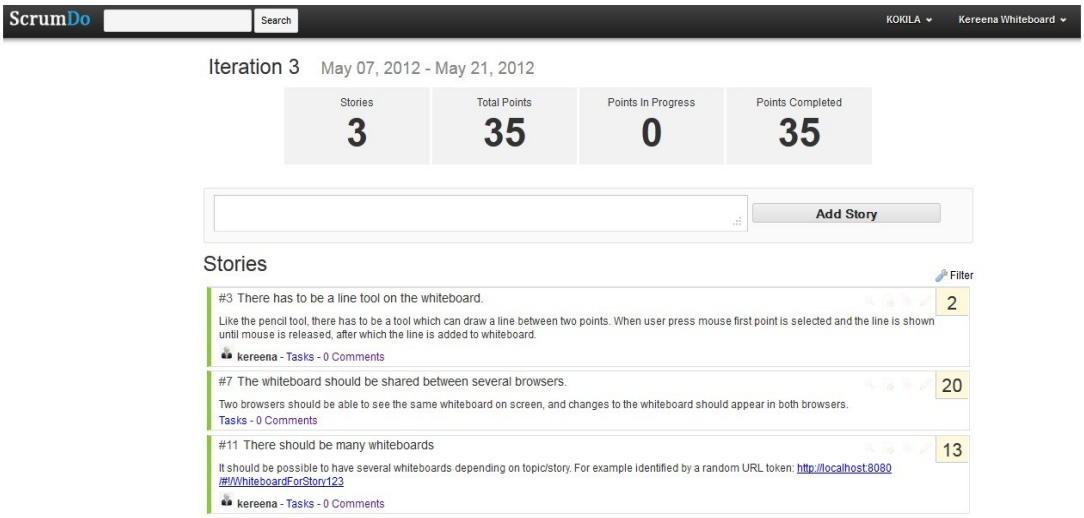
User stories


Add Story
Stories
Filter
#15 Identify technology for browser application
8
Need to select a technology to use for the client-side part of the whiteboard. Include libraries survey.
kereena - Tasks - 0 Comments
#16 Identify technology for server application
8
Select technology for server-side of the application. Include survey of useful libraries.
Tasks - 0 Comments
#17 Identify technology for database application
8
Identify technology to use for database.
Tasks - 0 Comments
#18 Create Google Docs document for project report
2
Remember to share it with READ permissions for reviewing KOKILA developers.
kereena - Tasks - 0 Comments
#19 Write Scrum survey in report
2
Tasks - 0 Comments
#20 Write India outsourcing survey in report.
2
Tasks - 0 Comments
#21 Identify problems related to Scrum and India outsourcing
13

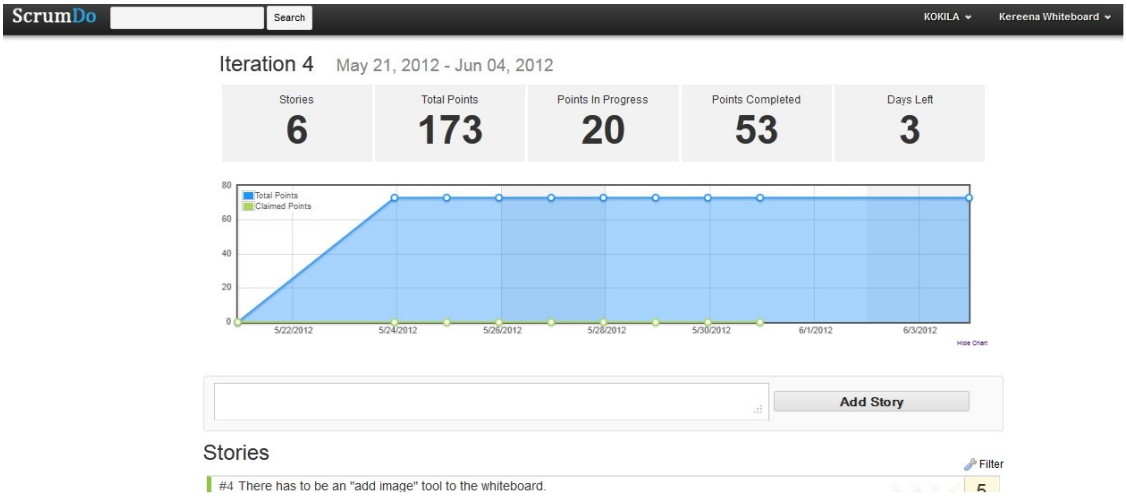
B.2 Iteration 2

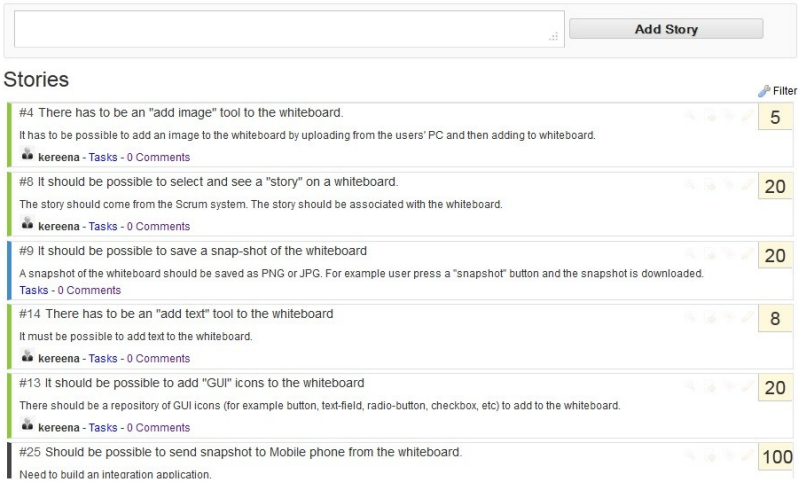


B.3 Iteration 3


ScrumDo
Search
KOKILA
Kereena Whiteboard
Iteration 3 May 07, 2012 - May 21, 2012
Stories 3
Total Points 35
Points In Progress 0
Points Completed 35
Add Story
Stories
Filter
#3 There has to be a line tool on the whiteboard. 2
Like the pencil tool, there has to be a tool which can draw a line between two points. When user press mouse first point is selected and the line is shown until mouse is released, after which the line is added to whiteboard.
kereena - Tasks - 0 Comments
#7 The whiteboard should be shared between several browsers. 20
Two browsers should be able to see the same whiteboard on screen, and changes to the whiteboard should appear in both browsers.
Tasks - 0 Comments
#11 There should be many whiteboards 13
It should be possible to have several whiteboards depending on topic/story. For example identified by a random URL token: http://localhost:8080/#!/WhiteboardForStory123
kereena - Tasks - 0 Comments

B.4 Iteration 4




Add Story
Stories
Filter
#4 There has to be an "add image" tool to the whiteboard.
5
It has to be possible to add an image to the whiteboard by uploading from the users' PC and then adding to whiteboard.
kereena - Tasks - 0 Comments
#8 It should be possible to select and see a "story" on a whiteboard.
20
The story should come from the Scrum system. The story should be associated with the whiteboard.
kereena - Tasks - 0 Comments
#9 It should be possible to save a snap-shot of the whiteboard
20
A snapshot of the whiteboard should be saved as PNG or JPG. For example user press a "snapshot" button and the snapshot is downloaded.
Tasks - 0 Comments
#14 There has to be an "add text" tool to the whiteboard
8
It must be possible to add text to the whiteboard.
kereena - Tasks - 0 Comments
#13 It should be possible to add "GUI" icons to the whiteboard
20
There should be a repository of GUI icons (for example button, text-field, radio-button, checkbox, etc) to add to the whiteboard.
kereena - Tasks - 0 Comments
#25 Should be possible to send snapshot to Mobile phone from the whiteboard.
100
Need to build an integration application.

Appendix C: User evaluation

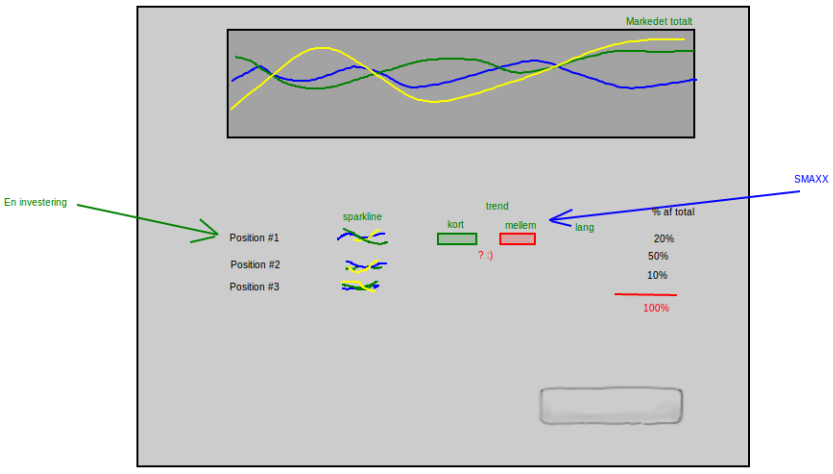
Product owner is PO, Developer is DEV.

Questions asked to evaluating users:
1) What was easy using the whiteboard?
2) What was hard or seemed strange using the whiteboard?
3) Any ideas for what could make it more useful?
4) Did the whiteboard help with your task?

C.1 Participants: PO: JDA (aktiehaj.dk), DEV: SPA (KOKILA)

This evaluation was concerned about discussing the elements of a new page to be constructed.

Resulting whiteboard:

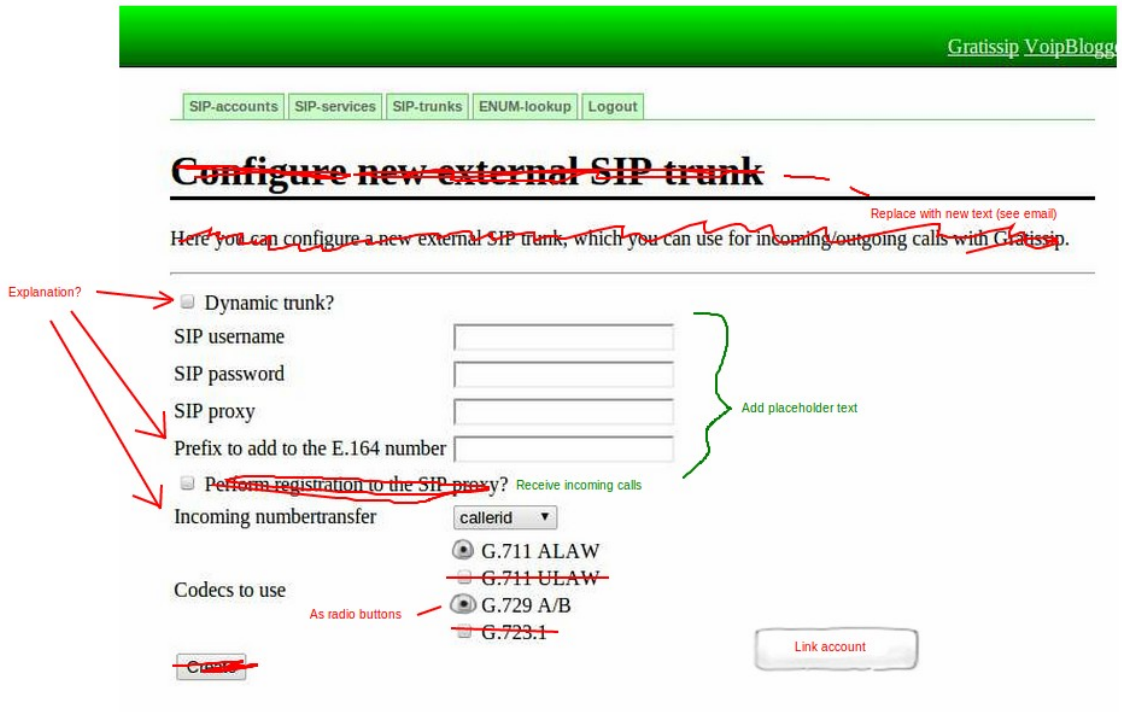


Answers from participants:
1) PO: It was good visualize what we were discussing. The whiteboard is simple and easy to get started with; DEV: It was easy to ask questions related to visualization.
2) PO: Could need an arrow function and adding text was a bit strange. Some more GUI components could be good when discussing GUIs. My screen was smaller than DEV's it was strange that he could draw "outside" of my screen; DEV: Same about arrow.
3) PO: Some wishes -- Support for touch devices (example IPad), it is more easy to draw with touch than mouse, ability to "point" to things for others; DEV: Same about pointer.
4) PO: Yes, but pen and paper + onsite would clearly be best; DEV: I think so too.

C.2 Participants: PO: RE (gratissip.dk), DEV: SPA (KOKILA)

This evaluation was concerned about discussing modifications to an existing web-site page.

Resulting whiteboard:



Answers from participants:
1) PO: It was nice to include a screenshot and discuss from it; DEV: It was nice to be able to give feedback during discussion. Normally only "screen-cast" without ability to add items. Improves my feedback and suggestions.
2) PO: Text tool was strange. Icon tool little strange but OK. Upload limit is annoying; DEV: Pointer would be nice.
3) PO: I would like to see a list of online users. Undo functionality is critical. The whiteboard is limited in size, an "unlimited" whiteboard could be useful. Ability to move already drawn items around; DEV: Agree. Maybe add some "chat" so I don't have to switch to Skype for it.

4) PO: I think so. It is good to have the whiteboard as documentation for what was discussed and we can return to it if there are still unclarities; DEV: Agree. It is easier than one-way communication. We can try out suggestions and argue for/agaist them.

Appendix D: User manual

This manual describes two sides of using: Firstly, how to get and start the application. Secondly, how to use the application.

Starting application

To start the application you need the following installed and ready:

1. Java JDK 1.6.x+ -- http://www.oracle.com/technetwork/java/index.html
2. Apache Maven -- http://maven.apache.org/download.html
3. MongoDB -- http://www.mongodb.org/downloads
4. Mercurial client -- http://tortoisehg.bitbucket.org/ (optional)

Also, your computer needs an internet connection, in order to download dependencies.
Once the above has been installed you are ready to continue to get the source code, in one of the following ways:

1. Copy the files from the CD that came with the report (this is done using your OSs copy and paste functionality).
2. “Clone” the files from the public Mercurial repository: https://bitbucket.org/kereena/whiteboard (this is done using the Mercurial client).

When the source code is ready, the application is built and started by Maven, using the following command (run from folder where “pom.xml” is found):

```
mvn compile exec:java -Dport=8080
```

The first time you build, it will download many dependencies and it can take some time. The following times, it will only check for updates, and then start the application. When the application is running, you should see a text like the following on your screen:

```
server started at http://127.0.0.1:8080/
```

This means the application is running and you can access it at http://127.0.0.1:8080/ using your Google Chrome browser.

If it is the first time you start the application, you can create a sample whiteboard in the database to work with by accessing the following URL:
http://127.0.0.1:8080/whiteboard/create?t=sample&o=sample&d=sample

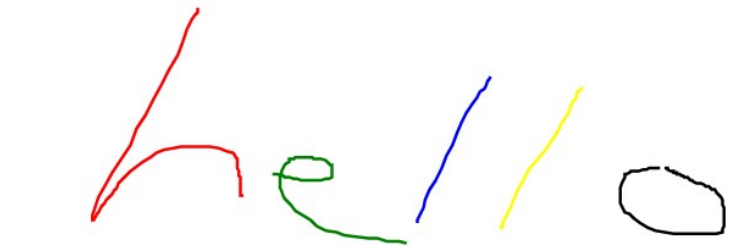
Using application

To use the application you need to load its web-interface in the browser, typically
http://127.0.0.1:8080/

You will see the welcome screen:

Select a whiteboard and click “join whiteboard”. If you wish to specify a username different from the auto-generated username, click on the username in the top-left corner (after Welcome) and enter your desired username.

After joining the whiteboard, you see the whiteboard screen:



If the whiteboard is already in use, you may see it has already contents. If there are other users online, you may also see that the contents of the whiteboard changes.

You can now modify the whiteboard with the selected tool. The selected tool is highlighted using a green background in the list of tools at the bottom right of the screen. By default the “pencil” tool is selected.

By clicking on one of the colors, your color changes, but the tool stays the same. The tools are from left to right:
Icon tool, Text tool, Pencil tool, Line tool, Rectangle tool, Image tool, Eraser tool, and Snapshot tool.